1988

A systolic array structure for on line system identification.

Tunc, Mumtaz.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A SYSTOLIC ARRAY STRUCTURE FOR ON LINE SYSTEM IDENTIFICATION

by

Mumtaz Tunc

December 1988

Thesis Advisor Roberto Cristi

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified		1b Restrictive Markings	
2a Security Classification Authority		3 Distribution Availability of Report: Approved for public release; distribution is unlimited.	
2b Declassification Downgrading Schedule			
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
6a Name of Performing Organization: Naval Postgraduate School	6b Office Symbol (*if applicable*) 32	7a Name of Monitoring Organization: Naval Postgraduate School	
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000		7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	
8a Name of Funding Sponsoring Organization	8b Office Symbol (*if applicable*)	9 Procurement Instrument Identification Number	
8c Address (*city, state, and ZIP code*)		10 Source of Funding Numbers	
		Program Element No | Project No | Task No | Work Unit Accession No	

11 Title (*include security classification*): A SYSTOLIC ARRAY STRUCTURE FOR ON LINE SYSTEM IDENTIFICATION

12 Personal Author(s): Mumtaz Tunc

13a Type of Report	13b Time Covered	14 Date of Report (*year, month, day*)	15 Page Count
Master's Thesis	From To	December 1988	82

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	Systolic Arrays, Recursive Least Squares Algorithm,Givens Rotation.

19 Abstract (*continue on reverse if necessary and identify by block number*)

In this study, we present an algorithm for system identification for systolic array implementation. With this schema, discrete samples of input and output data of a system with uncertain characteristics are used to determine the parameters of its model. The identification algorithm is based on recursive least squares, QR decomposition, and block processing techniques with covariance resetting. The identification process is based on the use of Givens rotation. Additionally, we want to address the following problems: how the round-off error propagates in time and the implementation in closed loop adaptive control. We will compare the implementation of fixed point arithmetic with the implementation of floating point arithmetic. This is primarily a theoretical investigation to be conducted with computer simulations where numerical results will be investigated.

20 Distribution Availability of Abstract		21 Abstract Security Classification	
☒ unclassified unlimited ☐ same as report ☐ DTIC users		Unclassified	
22a Name of Responsible Individual: Roberto Cristi		22b Telephone (*include Area code*): (408) 646-2223	22c Office Symbol: Code 62Cx

DD FORM 1473,84 MAR — 83 APR edition may be used until exhausted; all other editions are obsolete — security classification of this page

Approved for public release; distribution is unlimited.

A Systolic Array Structure for On Line System Identification

by

Mumtaz Tunc
LTJG, Turkish Navy
B.S., Turkish Naval Academy, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1988

ABSTRACT

In this study, we present an algorithm for system identification for systolic array implementation. With this schema, discrete samples of input and output data of a system with uncertain characteristics are used to determine the parameters of its model. The identification algorithm is based on recursive least squares, QR decomposition, and block processing techniques with covariance resetting. The identification process is based on the use of Givens rotation. Additionally, we want to address the following problems: how the round-off error propagates in time and the implementation in closed loop adaptive control. We will compare the implementation of fixed point arithmetic with the implementation of floating point arithmetic. This is primarily a theoretical investigation to be conducted with computer simulations where numerical results will be investigated.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

First of all, I wish to express my appreciation to the Turkish Navy Authority for the opportunity to study in the Naval Postgraduate School.

A significant debt of gratitude is owed to Professor Roberto Cristi, for the many hours of assistance and guidance he has extented, in the preparation of this thesis.

Also I would like to express my sincere appreciation to Professor C. H. Lee of the Department of Electrical and Computer Engineering of the Naval Postgraduate School, my second reader.

I. INTRODUCTION

A. BACKGROUND

When we design control systems we are faced with the problem of identifying the plant to be controlled. In particular, we try to determine the parameters of a mathematical model (i.e., a linear differential or difference equation) which best fits the input-output data of the given plant.

In some cases, insight of a model may be obtained from the laws of Physics, Chemistry, etc. In most cases, this may not be possible due to the complexity of the physical factors involved. In these instances, it may be possible to derive the values of the parameters by observing the nature of the system's response under appropriate experimental conditions. This procedure is called "parameter estimation".

The problem of adaptively controlling systems with uncertain characteristics depends on identification of the unknown system parameters. In these cases, the parameters of the controller are computed on the basis of the current estimate of the dynamics. Figure 1 shows a general structure of an adaptive control system.

The purpose of parameter estimation is to best fit a proper model to the input-output data of the system under investigation. The main issues are the following:

1. Select an appropriate class of models, and
2. Select an appropriate estimation algorithm.

For the particular case of estimating the parameters of linear models, we search within the class of models with a given fixed order for the one which minimizes a prediction error criterion.

Firstly, suppose we wish to select a model of a system based on input-output measurements, in the form

$$y(t) = \varphi^T(t)\theta + v(t) \tag{1.1}$$

where $y(t)$ and $\varphi^T(t)$ are determined on the basis of the output and input signals and θ is an array of unknown parameters to be determinated. The term $v(t)$ represents noise or other modeling errors. Further information about this equation will be given in the next chapter. This class of linear models is preferable because of its simplicity and the amount of theory developed to analyze them.

Secondly, we need to select an appropriate estimation algorithm. Although a wide choice exists, the most effective in terms of speed of convergence and accuracy is the recursive least squares algorithm [Ref. 1].

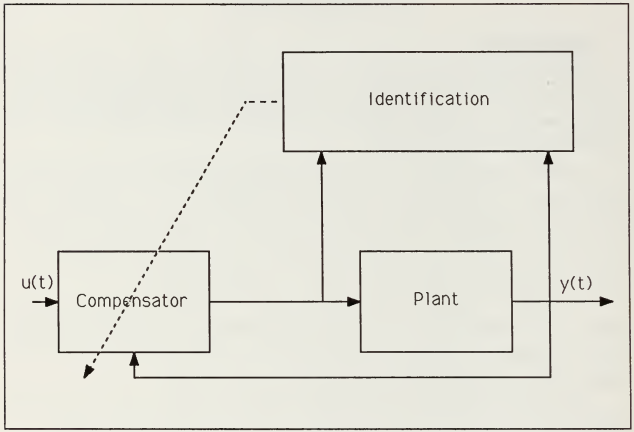


Figure 1. The Adaptive Control System.

The origins of the least-squares method can be traced back to Gauss as in [Ref. 2]. In its recursive version it has been formulated by several authors. The major drawback of recursive least-squares identification is its cost in terms of complexity, which might make it unsuitable for real time identification of a large number of parameters, since the size of the matrices involved grow with the complexity of the the system to be estimated. Although available microprocessors are effective for low order systems and slow sampling rates, more complex problems require improved capabilities.

In this thesis we address the problem of implementing recursive least-squares identification using parallel processing techniques and systolic arrays. Particularly, the parameter estimates are determined by the least-squares solution of a redundant number of linear equations obtained from the measured data. The techniques of solutions for this class of equations are based on QR decomposition, discussed in the next chapter.

B. SYSTOLIC ARRAY

The idea of systolic array was developed by Kung and associates [Ref. 3] at Carnegie-Mellon University, and many versions of systolic processors are being designed by universities and industrial organizations. This subsection reviews the basic principle of systolic architectures and explains why they should result in cost-effective high performance, special purpose systems for a wide range of potential applications.

A systolic array consists of a set of interconnected cells, each capable of performing simple operations. Since simple, regular communication and control structures have substantial advantages over complicated ones in design and implementation, the cells in a systolic system are typically interconnected with the immediate neighbors. In the systolic array we are considering there are two kinds of cells (boundary cell, internal cell). Each cell in the array is provided with local memory of its own, and it is connected only to its nearest neighbors. The array is designed such that regular streams of data are clocked through it in a highly rhythmic fashion, much like the pumping action of the human heart; hence the name "systolic". Information in a systolic system flows between cells in a pipelined fashion, and communication with the outside world occurs only at the boundary cells.

Basic principle of a systolic array is illustrated in Figure 2. Suppose each processing element in Figure 2 operates with a clock period of 100 ns. The conventional memory and processor organization in Figure 2a has 5 million operation per second. With same clock rate, the systolic array will result in 30 million operation per second performance provided the processing elements operate in parallel on pipelined data. The gain in processing speed has been increased six times. Being able to use each input data item a number of times is just one of the many advantages of the systolic approach. Other advantages include modular expandability, simple and regular data and control flows, use of simple and uniform cells and fast response time.

Previous authors have presented parameter estimation algorithms using systolic arrays. The general idea has been to solve a system of linear equations in two stages:

1. Triangularization of the matrix of coefficients,
2. Solving by successive substition.

As explained in Chapter III the previous algorithms used a triangular systolic array to triangularize the matrix, and a linear systolic array configuration to solve for the parameters. The linear section requires operations such as divisions which are hard to implement by simple processor operations. Because of this, a new algorithm was

developed. In our implementation, a second identical triangular array has been used instead of the linear systolic array. It is characterized by the fact that only orthogonal operations are involved, making the algorithm numerically more stable and easily implementable by simple shift and add operations. Also this algorithm needs two different cells (boundary and internal); previous algorithms needed four different cells (two for a triangular array, two for a linear array). Although more total cells are required in this implementation, the cost of additional cells in a VLSI scheme is considered to be minimal.

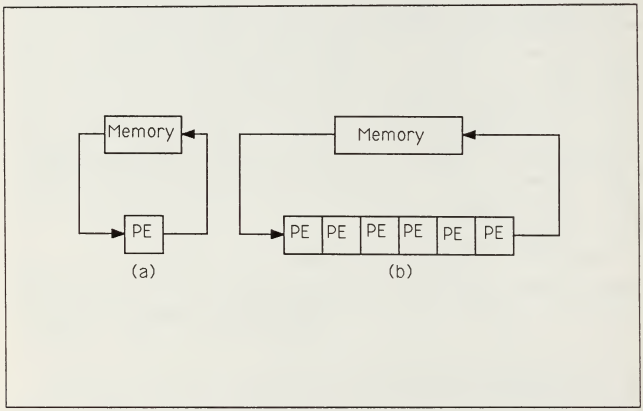


Figure 2. The Concept of Systolic Processor Array.

In implementations based on fixed point arithmetics, vector rotations necessary for the QR factorization can be implemented by a CORDIC algorithm, based on simple shift, add operations.

The use of fixed point versus floating point arithmetic is considered during this investigation. Because fixed point operations are based on simple shift functions and finite registers, which are simple to implement, it seems advantageous to use fixed point values. However, since input and output data do not naturally appear as integer values, there is concern over loss of accuracy due to necessary scaling and truncation.

This research report is divided as follows: Chapter II discusses the methods of system identification, i.e., solution of systems of linear equations, QR decomposition, and recursive least squares algorithm, block processing and covariance resetting. Chapter III discusses the Givens rotation, the CORDIC technique and implementation of the systolic arrays. Chapter IV presents the simulation results, and Chapter V shows the final conclusions. A listing of the computer program is used to simulate the systolic arrays is found in the Appendix B.

II. MODELS FOR SYSTEM IDENTIFICATION

A. LINEAR SYSTEM MODELING

Suppose we wish to identify a model of a system based on input-output measurements. As shown in Figure 3 consider a system with a single input u(t) and single output y(t).

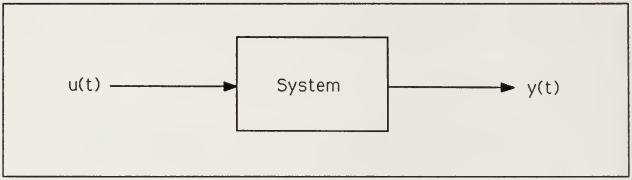


Figure 3. Simple Linear System.

If we consider a linear difference equation to model the dynamics, we can write

$$y(t) + a_1 y(t-1) + ... + a_n y(t-n) = b_1 u(t-1) + ... + b_m u(t-m) + v(t) \tag{2.1}$$

or, equivalently

$$y(t) = -a_1 y(t-1) - .... - a_n y(t-n) + b_1 u(t-1) + ... + b_m u(t-m) + v(t) \tag{2.2}$$

where v(t) shows noise or other modeling errors, a_i's and b_i's are real constants, and the equation is of nth order. Since the output depends not only on the input but on the previous output values, this discrete system is known as *recursive system.* Equation (2.2) can be written in the matrix form

$$y(t) = [a_1,..., a_n, b_1,..., b_m] \begin{bmatrix} -y(t-1) \\ . \\ -y(t-n) \\ u(t-1) \\ . \\ u(t-m) \end{bmatrix} \tag{2.3}$$

where θ represents the parameter vector

$$\theta^T = [a_1, \ldots, a_n, b_1, \ldots, b_m] \tag{2.4}$$

and $\varphi(t)$ represents the regression vector.

$$\varphi(t) = \begin{bmatrix} -y(t-1) \\ \cdot \\ -y(t-n) \\ u(t-1) \\ \cdot \\ u(t-m) \end{bmatrix} \tag{2.5}$$

By the above definitions we can write Equation (2.1) as

$$y(t) = \varphi^T(t)\theta + v(t) \tag{2.6}$$

The problem now is to estimate the parameter vector θ from measurements of the sequences y(t) and $\varphi(t)$. Normally, if the number of samples of u(t) and y(t) (i.e., the number of equations) equals the number of unknowns (m+n), we can solve exactly for θ. However, the number of equations is usually greater than the number of unknowns, since we tend to collect a large amount of data. For this reason, this system of equations in general does not have a solution. Ideally, in the noiseless case (v(t)=0) a solution exists, regardless of the number of equations. However, since noise and numerical errors are present, we look for the least squares solution of the system of equations, i.e., for the solution which minimizes the error.

If we take N samples of Equation (2.6) and do not consider noise or other modeling errors we can write the system of N equations in matrix form as

$$\begin{bmatrix} y(t) \\ y(t-1) \\ \cdot \\ \cdot \\ y(t-N) \end{bmatrix} = \begin{bmatrix} \varphi^T(t) \\ \varphi^T(t-1) \\ \cdot \\ \cdot \\ \varphi^T(t-N) \end{bmatrix} \theta \tag{2.7}$$

or, equivalently

$$\underline{b} = A\theta \tag{2.8}$$

where $A \in \mathcal{R}^{M \times N}$, $\theta \in \mathcal{R}^{N}$, $\underline{b} \in \mathcal{R}^{M}$. When N = M and A is full rank and invertible, we can solve uniquely for θ as

$$\theta = A^{-1}\underline{b} \tag{2.9}$$

However, in signal processing applications we often face the case of M > N (i.e., more equations than unknowns), and the solution of (2.9) is defined in the least square sense by minimization of the error

$$\varepsilon(\theta) = \|A\theta - \underline{b}\|^2 \tag{2.10}$$

Therefore, the least squares solution θ_s of (2.7) is implicitely defined as

$$\|A\theta_s - \underline{b}\|^2 = \min_{\theta}\|A\theta - \underline{b}\|^2 \tag{2.11}$$

The least squares solution always exists, althought it might not be unique. We can solve Equation (2.11) by pseudoinverse which however, involves matrix inversion. Another method is to triangularize A in Equation (2.8) and solve for θ by successive back substitutions.

When M > N we meet the dilemma of triangularizing an array. The solution is to triangularize the upper part of the A matrix, leaving one or more rows of zeros at the bottom as a result. This will allow us to solve for θ in the *least square sense* as indicated in Equation (2.11). This is known as QR decomposition.

B. SOLUTION OF THE LEAST-SQUARES PROBLEM USING QR DECOMPOSITION

A numerically attractive method for determing the least squares solution of a system of linear equations is provided by the QR decomposition of a matrix [Ref. 4]. Consider again Equation (2.8) with M > N. It can be shown that we can always factorize A as

$$A = QR \tag{2.12}$$

where A is a M × N matrix, Q is an M × M orthogonal matrix such that $Q^TQ = I$ and R is defined as

$$R = \left| \begin{array}{c} \underline{R} \\ --- \\ 0 \end{array} \right| \tag{2.13}$$

where $\underline{R}$ is an $N \times N$ upper triangular matrix and 0 represents null matrix. It is not restrictive to assume the diagonal entries of $\underline{R}$ to be non negative. Now we write again Equation (2.9)

$$A\theta = \underline{b}$$

with the equality intended in least squares sense, and multiply both sides by Q^T

$$Q^T Q R\theta = Q^T \underline{b}$$

since $Q^T Q = I$

$$R\theta = Q^T \underline{b}$$

and, therefore,

$$\left| \begin{array}{c} \underline{R} \\ --- \\ 0 \end{array} \right| \theta = Q^T \underline{b} = \left| \begin{array}{c} \beta_1 \\ \beta_2 \end{array} \right| \tag{2.14}$$

It is now easy to see that the error $\varepsilon(\theta)$ is given by

$$\|A\theta - \underline{b}\|^2 = \|\underline{R}\theta - \beta_1\|^2 + \|\beta_2\|^2 \tag{2.15}$$

which is minimal when

$$\underline{R}\theta = \beta_1 \tag{2.16}$$

since β_2 is independent of θ. In particular, if A is full rank and $M > N$, then the solution is unique, $\underline{R}$ is invertible and we can compute the least squares solution of (2.7) and (2.8) as

$$\theta_s = \underline{R}^{-1} \beta_1$$

Now the solution of Equation (2.8) in the least square sense can be computed in the same manner as the solution of a system of N equations and N unknowns in Equation (2.16).

There is a number of methods available that can be used to compute the upper triangular matrix R (e.g., Householder transformation, Gram-Schmidt procedure, Givens rotation). We will concentrate on Givens rotation. The Givens rotation is an attractive method for systolic array implementation since it successively manipulates two adjacent rows of the matrix at a time. CORDIC techniques can be used to implement these rotations and are discussed in the next chapter.

C. RECURSIVE LEAST SQUARES ALGORITHM

It is possible to estimate the values of the model parameters θ by observing the nature of the system's response under appropriorite experimental conditions. The idea for the recursive identification problem is to compare the output with that of an adjustable model, and update the parameters until the error between the outputs of the model and the system is minimized as in Figure 4.

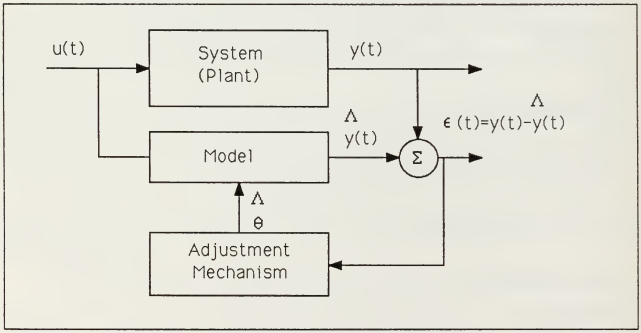


Figure 4. The Minimization of Error in the System.

As seen in the previous section, assumption of a linear time invariant model leads to the relation

$$y(t) = \varphi^T(t)\theta \tag{2.17}$$

with θ the parameters to be estimated, and $\varphi(t)$ a measurable sequence. A recursive least squares procedure can be determined by defining $\hat{\theta}_t$ the estimate of θ at time t, as the vector which minimizes

$$\varepsilon_t^2(\theta) = \sum_{\tau=0}^{t-1} \left| y(\tau) - \varphi^T(\tau)\theta \right|^2 \tag{2.18}$$

Particularly, $\hat{\theta}_t$ is the least squares solution of the equations

$$\begin{vmatrix} \phi^T(t-1) \\ \cdot \\ \cdot \\ \phi^T(0) \end{vmatrix} \hat{\theta}_t = \begin{vmatrix} y(t-1) \\ \cdot \\ \cdot \\ y(0) \end{vmatrix} \quad \rightarrow \quad A(t-1)\hat{\theta}_t = y(t-1) \tag{2.19}$$

It is possible to show that $\hat{\theta}_t$ can be recursively computed as [Ref. 2]

$$\hat{\theta}_{t+1} = \hat{\theta}_t + P(t-1)\frac{\varphi(t)(y(t) - \hat{\theta}_t^T\varphi(t))}{1 + \varphi^T(t)P(t-1)\varphi(t)} \tag{2.20}$$

where $P(t) = (A^T(t)A(t))^{-1}$ satisfies the recursion

$$P(t) = P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{1 + \varphi^T(t)P(t-1)\varphi(t)} \tag{2.21}$$

So far, we have neglected the problem of existence of the solution. The matrix A(t) can be singular, and then P(t) might not exist. Furthermore, we can not initialize A(-1) = 0 because, in this case this would yield P(-1) undefined, and we would not know how to start the sequence P(t). A solution to this problem is to incorporate initial conditions into A(t) so that P(t) can be computed at each t. Therefore, we modify the definition of $\hat{\theta}_t$ in Equation (2.19), so that initial conditions of the matrix P(t) can be accounted. Let $A(-1)$ be any arbitrary matrix (it must be full rank). For convenience let $A(-1)$ be square, with $\det(A(-1)) \neq 0$, and define $P(-1) = (A^T(-1)A(-1))^{-1}$. This can be done by defining a new error criterion

$$\varepsilon_t^2(\theta) = \sum_{\tau=0}^{t-1} \left| y(\tau) - \theta^T\varphi(\tau) \right|^2 + (\theta - \hat{\theta}_0)^T(A^T(-1)A(-1))^{-1}(\theta - \hat{\theta}_0) \tag{2.22}$$

which takes the initial estimate $\hat{\theta}_0$ into account. This new formulation leads to the matrix A(t) as

$$A(t) = \begin{vmatrix} \phi^T(t) \\ \phi^T(t-1) \\ \cdot \\ \phi^T(0) \\ \text{------} \\ A(-1) \end{vmatrix} \tag{2.23}$$

We choose A(-1) = $\sigma_0 I$, where $\sigma_0 > 0$ is some arbitrary constant, and I the identity matrix of appropriate dimensions. In this way the matrix P(t) is always defined since $A^T(t)A(t)$ is never singular. Then, by algebraic manipulations, Equation (2.19) can be written as

$$\begin{vmatrix} \phi^T(t-1) \\ \cdot \\ \cdot \\ \phi^T(0) \\ \text{------} \\ A(-1) \end{vmatrix} \hat{\theta}_t = \begin{vmatrix} y(t-1) \\ \cdot \\ \cdot \\ y(0) \\ \text{------} \\ A(-1)\hat{\theta}_0 \end{vmatrix} \tag{2.24}$$

The solution to (2.24) is computed recursively using (2.17) and (2.18) and the appropriate initial conditions.

D. BLOCK PROCESSING AND COVARIANCE RESETTING

From the previous considerations, the algorithm was described that allows us to estimate the parameter θ recursively. By using Equation (2.20) and $P(t) = (A(t)^T A(t))^{-1}$ we obtain

$$P(t) = (A^T(t)A(t))^{-1} = (\sigma_0^2 I + \sum_{i=0}^{t} \varphi(i)\varphi^T(i))^{-1} \tag{2.25}$$

In general, the term $\sum_{i=0}^{t} \varphi(i)\varphi^T(i)$ is likely to grow with time, so that P(t) $\rightarrow$ 0 as t $\rightarrow \infty$ [Ref. 2]. Therefore, the algorithm loses sensitivity as t increases, and later values of θ

may not be as accurate as earlier values, especially if our model changes with time. There are two possible solutions to this problem:

1. The use of a "forgetting factor", and
2. The covariance resetting approach.

By the forgetting factor approach we minimize the error

$$\|\varepsilon(t)\|^2 = \sum_{k=0}^{t} \lambda^{t-k}(y(k) - \phi^T(k)\hat{\theta}_{t+1})^2 + \sigma_0^2\left\|\hat{\theta}_{t+1} - \theta_0\right\|^2 \tag{2.26}$$

where $0 < \lambda < 1$ is the forgetting factor. This has the effect of assigning a higher weight to more recent data.

The covariance resetting approach, on the other hand, divides the time scale into segments of equal and fixed length N as in Figure 5. At the end of each time block,

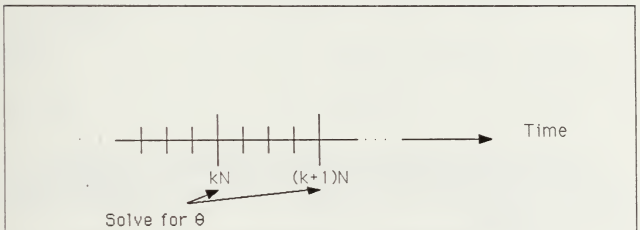


Figure 5. Covariance Resetting Approach.

we reset the covariance matrix P(t). Although it can be reset at any time, for convenience we choose the end of each block, and P(t) now becomes

$$P(t) = \begin{Bmatrix} \sigma_0^{-1} I & t = kN - 1 \quad k = 0,1,2,\ldots \\ Equation(2.21) & otherwise \end{Bmatrix} \tag{2.27}$$

In particular, at the beginning of each interval the systolic array is initialized as $\sigma_0 I$, and at the end of the interval (at kN + N-1 for the kth interval) the data are exited from the array and the system is solved to obtain $\hat{\theta}_{(k-1)N}$. This estimate is used as an initial condition for the next time block.

In an adaptive control context it has been shown [Ref. 5] that an external input u(t), sufficiently rich in frequency (n sinusoids), together with blocks I_k of sufficent length N provides a guarantee for a consistent estimation as

$$\theta \rightarrow \theta^{\times}$$

where $\theta^{\times}$ represents actual parameters. The effect of various lengths of N will be investigated later.

If we apply the considerations given to the general case, we can see that the parameter estimates at the end of each time block are related by the equation

$$\begin{vmatrix} \phi^T((k+1)N-1) \\ \cdot \\ \cdot \\ \phi^T(kN) \\ \text{-----} \\ \sigma_0 I \end{vmatrix} \hat{\theta}_{(k+1)N} = \begin{vmatrix} y((k+1)N-1) \\ \cdot \\ \cdot \\ y(kN) \\ \text{-----} \\ \sigma_0 \hat{\theta}_{kN} \end{vmatrix} \tag{2.28}$$

The systolic array implementation is based on this equation. In particular, $\hat{\theta}_{(k-1)N}$ is computed from Equation (2.28) at the end of each time block by making the leftmost matrix upper triangular.

Although, the algorithm presented in this Chapter assumes a single input, single output (SISO) plant, it can be extended to the multiple input-output (MIMO) plant [Ref. 6]. Additionally, we assume the plant to be causal and of known order.

III. SYSTOLIC ARRAY IMPLEMENTATION

Now, the problem is to compute the solution of Equation (2.28) using systolic arrays. In particular, Equation (2.28) is appropriate for parallel implementation. As described in [Ref. 5], the identification problem can be constructed again into a set of linear equations as

$$R_k \theta_{(k+1)N} = \beta_{k1} \tag{3.1}$$

where R_k is in upper triangular form. We can compute $\theta_{(k+1)N}$ by using two processors in cascade: one to compute R_k and β_{k1} , and the second to compute θ from Equation (3.1). Notice that Equation (3.1) does not require any explicit matrix inversion since R_k is in upper triangular form.

As seen in Equation (2.28), we initialize the array at the beginning of each time block such that

$$R_0 = \sigma_0 I$$

$$\beta_0 = \sigma_0 \bar{\theta}_{k,N}$$

This has the effect of initializing the R_k matrix in (3.1) to an upper triangular form. Then, at each discrete time t, an array of data $\varphi(t)$ and y(t) will be passed to the systolic array. The task of the array is to re-triangularize the data at each clock pulse, so the matrix remains in the form prescribed by (3.1). It is then a simple matter to solve for $\theta_{(k+1)N}$. This technique is based on QR decomposition as the means to triangularize the data array.

The value of σ_0 relates to the confidence we have in the initial estimates of $\hat{\theta}_0$. As seen in Equation (2.26) a larger value of σ_0 , will increase the confidence on the initial estimate of $\hat{\theta}_0$. Equations (2.24-2.28) show the role that σ_0 plays.

A. GIVENS ROTATION

The orthogonal triangularization process may be carried out by using various techniques. One of the techniques is the Gram-Schmidt orthogonalization procedure that provides the mathematical basis for the pipelined lattice predictor. Another powerful technique of triangularization by QR decomposition is provided by the Givens

rotation. The reason for this choice is the fact that the Givens rotation operates on pairs of adjacent rows, making it suitable for systolic array implementation. The object is to combine two adjacent rows in the matrix, forcing zeros in the appropriate positions so to obtain an upper triangular data matrix. Figure 6 shows an example of triangularization by successive Givens rotations.

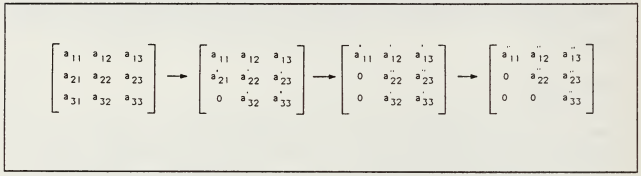


Figure 6. Example of Row Operations.

Basic idea: Consider any two rows of data as

$$\begin{vmatrix} a_{k,1} & a_{k,2} & \cdots\cdots & a_{k,n} \\ a_{k+1,1} & a_{k+1,2} & \cdots\cdots & a_{k+1,n} \end{vmatrix}$$

We can see these two rows as a sequence of vectors in Figure 7.

If we rotate all these vectors by an appropriate angle α then $a_{k+1,1} \rightarrow 0$ since the leftmost vector becomes parallel to the horizontal axis. This is accomplished by the following operation

$$\begin{vmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{vmatrix} \begin{vmatrix} a_{k,1} & a_{k,2} & \cdots\cdots & a_{k,n} \\ a_{k+1,1} & a_{k+1,2} & \cdots\cdots & a_{k+1,n} \end{vmatrix} = \begin{vmatrix} a'_{k,1} & a'_{k,2} & \cdots\cdots & a'_{k,n} \\ 0 & a'_{k+1,2} & \cdots\cdots & a'_{k+1,n} \end{vmatrix}$$

The value of the angle α can be determined using simple trigonometry

$$\cos\alpha = \frac{a_{k,1}}{\sqrt{a_{k,1}^2 + a_{k+1,1}^2}} \qquad \sin\alpha = \frac{a_{k+1,1}}{\sqrt{a_{k,1}^2 + a_{k+1,1}^2}} \tag{3.2}$$

This same rotation α is then applied to the remaining (x,y) vectors in the affected rows (i.e., rows k+1, k) to ensure consistency. Next, the sequence of rotations is repeated for the remaining pairs of rows (i.e., rows k+n, k+n-1; k+n-1, k+n-2;...; k+2, k+1) in order to force zeros in the correct locations that leave an upper triangular matrix.

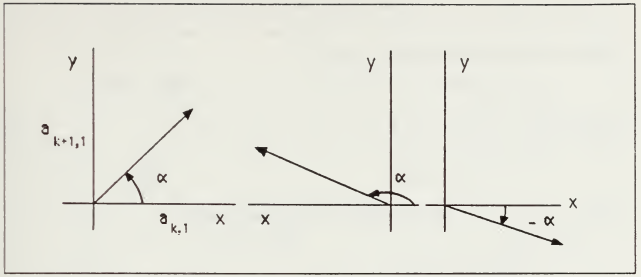


Figure 7. Example of Vector Operations.

We remember that the data matrix is initialized to an upper triangular form ($\sigma_0 I$). Then, as we take samples from the signals in the plant, new values of φ (t) and y(t) are added to the matrix. By a sequence of Givens rotations, we can transform it into an upper triangular matrix. An example set of rotations is shown in Figure 8, where the operations performed by the systolic array at each clock pulse are illustrated. This is repeated until t = kN (i.e., at the end of the time block), at which time the parameters θ are solved for, the matrix is reinitialized, and the process is repeated.

Basis of the Givens rotation is the matrix

$$Q(p,q) = \begin{vmatrix} I_1 & 0 & 0 \\ 0 & r(p,q) & 0 \\ 0 & 0 & I_2 \end{vmatrix} \tag{3.3}$$

associated to each pair of indexes p, q $\in$ (1,n+1), with I_1 and I_2 identity matrices of dimensions (q-2) $\times$ (q-2) and (n+1-q) $\times$ (n+1-q) respectively, and r is a 2 $\times$ 2 matrix of the form

$$r(p,q) = \begin{vmatrix} c(p,q) & s(p,q) \\ -s(p,q) & c(p,q) \end{vmatrix} \tag{3.4}$$

The matrix $Q(p,q)$ is an orthogonal matrix and has the property that it affects only two rows at a time, i.e., rows q and q-1. Application of the transformation $Q(p,q)$ to any matrix A of appropriate dimensions implies

$$Q(p,q)A = \begin{vmatrix} x & x & x \\ x & z & x \\ x & 0 & x \\ x & x & x \end{vmatrix} \tag{3.5}$$

where x indicates other elements of the matrix and z indicates

$$z = \sqrt{a_{q-1,p}^2 + a_{q,p}^2}$$

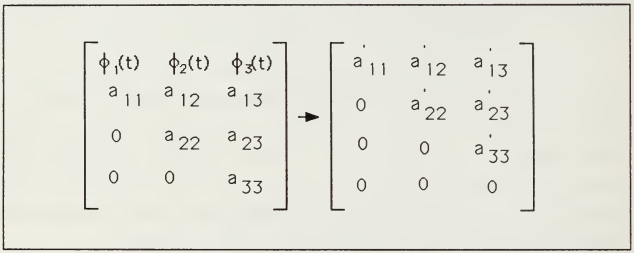


Figure 8. Example of Givens Rotation.

In Equation (3.4) c(p,q) and s(p,q) are provided such that

$$\begin{aligned} c(p,q)a_{q-1,p} + s(p,q)a_{q,p} &= z \\ c(p,q)a_{q,p} - s(p,q)a_{q-1,p} &= 0 \end{aligned} \tag{3.6}$$

In Figure 8, an example of application of the Givens rotation Q(3,4)Q(2,3)Q(1,2) to the left matrix results in the rotated matrix shown on the right [Ref. 7]. As seen before the Givens rotation requires addition, substraction, multiplication, division, and squares. Figure 9 illustrates the operations of each of the cells. Next, we will see how to perform the rotations, by using add and shift operations only in the CORDIC technique algorithm.

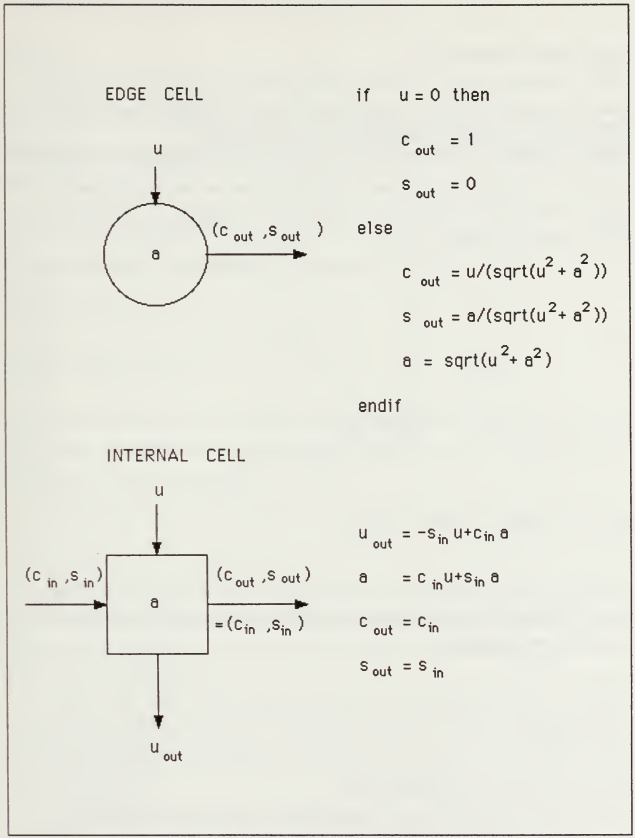


Figure 9. Givens Rotation Algorithm.

B. THE CORDIC TECHNIQUE

Here is a simple, accurate and relatively fast method for generating trigonometric functions. The technique proposed is based on the ingenious *coordinate rotation digital computer* (CORDIC) technique developed by J. E. Volder [Ref. 8].

The principle involved in CORDIC is to rotate a vector, represented by its components X and Y, through a set of successive elementary rotations as seen in Figure 10. Each single rotation of the vector (X,Y) is computed at each step by a combination of simple add, subtract, and shift operations.

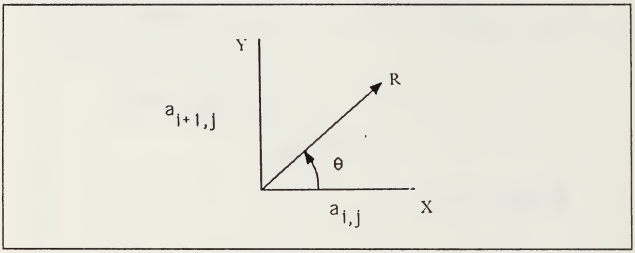


Figure 10. Example of CORDIC Rotation.

Consider a vector R represented by its components X and Y and rotate it through an angle $\pm\theta$. The results are the rotated coordinates X' and Y', which are related to X and Y by the following equations

$$X' = X\cos\theta \pm Y\sin\theta$$
$$Y' = Y\cos\theta \mp X\sin\theta \tag{3.7}$$

where the top sign refers to clockwise rotation.

The CORDIC principle is based on performing a vector rotation in a sequence of angular steps α_i such that the sum of them equals to θ, that is

$$\theta = \alpha_0 \pm \alpha_1 \pm \alpha_2 \ldots\ldots \pm \alpha_n \tag{3.8}$$

If we define $\gamma = \pm 1$, then we can express θ as

$$\theta = \sum_{i=0}^{n} \gamma_i \alpha_i \tag{3.9}$$

The problem is to choose the value of α_i such that the equations above can be implemented with simple add and shift operations. This is possible if the values are chosen such that

$$\alpha_i = \tan^{-1}(2^{-i}) \qquad i = 0,1,2....n \tag{3.10}$$

Dividing both sides of Equation (3.7) by $\cos \theta_i$ gives

$$\begin{aligned} \frac{X'_i}{\cos \theta_i} &= X_i \pm Y_i \tan \theta_i = X_{i+1} \\ \frac{Y'_i}{\cos \theta_i} &= Y_i \pm X_i \tan \theta_i = Y_{i+1} \end{aligned} \tag{3.11}$$

The factors $X'_i / \cos \theta_i$ and $Y'_i / \cos \theta_i$ are the rotated components of the vector (X,Y). The new vector is not only rotated, but also scaled by a factor $1/\cos \theta_i$. If θ is now replaced by $\alpha_i = \tan^{-1}(2^{-i})$ the new equations

$$\begin{aligned} X_{i+1} &= X_i \pm Y_i 2^{-i} = X_i + \gamma_i Y_i 2^{-i} = K_i X' \\ Y_{i+1} &= Y_i \mp X_i 2^{-i} = Y_i - \gamma_i X_i 2^{-i} = K_i Y' \end{aligned} \tag{3.12}$$

The terms X_i and Y_i indicate the components of the initial vector R. The term K_i is the factor $1/\cos \theta_i$ by which X_{i+1} and Y_{i+1} are larger than the components X' and Y' of a vector which is only rotated.

Rotations by angles smaller than 90°, the index (i) starts with 0,1,2,.... and continues to n. The resulting values for 2^{-i} , α_i and K_i are given in Table 1. During each rotation the magnitude of R increases by $K_i = 1/\cos \alpha_i$. After n rotation steps the value of K_n becomes

$$K_n = \prod_{i=0}^{i=n} K_i = \frac{1}{\cos \alpha_0} \cdot \frac{1}{\cos \alpha_1} \cdot \frac{1}{\cos \alpha_2} \cdots \frac{1}{\cos \alpha_n} \tag{3.13}$$

The value of K_n thus increases with each rotation step and approaches 1.6468 for high values of i.

Table 1. THE BINARY CORDIC CONSTANTS.

i	2^{-i}	α_i (in degrees)	K_i
0	1.0000000000	45.00000000	1.414213500
1	0.5000000000	26.56505124	1.581138826
2	0.2500000000	14.03624340	1.629800596
3	0.1250000000	7.12501632	1.642484060
4	0.0625000000	3.57633432	1.645688908
5	0.0312500000	1.78991064	1.646492240
6	0.0156250000	0.89517384	1.646639215
7	0.0078125000	0.44761428	1.646743467
8	0.0039062500	0.22381056	1.646756030
9	0.0019531250	0.11190564	1.646759170
10	0.0009765625	0.05595300	1.646759954

Since we assume the number of rotations to be constant, K_n is also constant. And to correct for the encrease of the magnitude of R, the resulting X_n and Y_n values must be divided by K_n after the rotation operation is completed. Figure 11 illustrates a series of rotations for an arbitrary vector which we desire to rotate by 40° . Notice that in the figure, the desired angular value is approached quickly, because α_i decreases by approximately half during each step. The γ sequence in the figure would be (+1, -1, +1, +1, +1, -1, -1, +1, -1, +1).

In an ideal case such as the one presented, the value of Y decreases during each step. However, this may not always be the case. For example, consider the vector (X,Y) = (5,2) and α = 12.5°. The first rotation in the CORDIC algorithm would be −45°, this gives us a rotated vector (X,Y) = (4.94,-2.12) and α = −32.5°. Notice that the value of $|Y|$ has actually increased. To make the algorithm more efficent, we modify the sequence γ to include the value zero. Whenever a rotation causes $|Y|$ to increase, we do

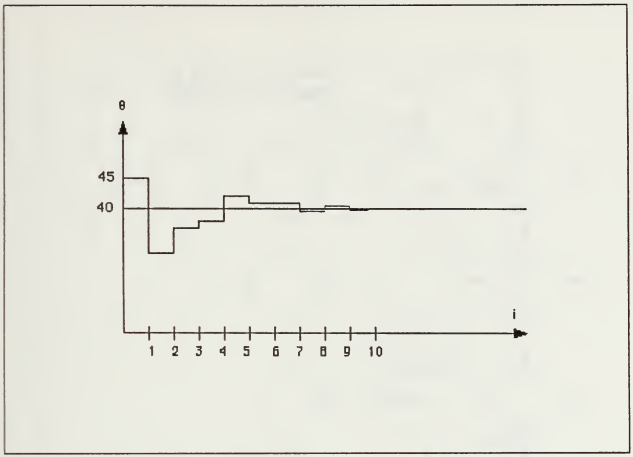


Figure 11. Set of CORDIC Rotations.

not perform it, and we set the corresponding γ_i to zero. Then, we continue on with the next rotation value and repeat the process. For the example just mentioned, the next value to be tried would be 26.6° . A CORDIC rotation algorithm can be seen in Figure 12.

C. SYSTOLIC ARRAYS

In this section we will examine the parallel structure that will be used to solve the least squares algorithm described above. In particular we aim at a structure that accepts a sequence of regression vectors $\varphi(n)$ and signal y(n) as input and then outputs the estimate for the parameter θ . Specifically, we are interested in a high performance parallel structure that can be implemented directly as a hardware device in order to deliver maximum throughput. Systolic arrays represent a structure suitable for these characteristics.

```
Initialize: If x > 0 then x = x, y = y
                    else x =-x, y =-y
if y > 0 then σ0 = +1
         else σ0 = -1
i=0
while i > N-1 do
if y_i - σ_i 2^-i x_i > y_i then
x_i+1 := x_i
y_i+1 := y_i
σ_i+1 := 0
else
x_i+1 := x_i + σ_i 2^-i y_i
y_i+1 := y_i - σ_i 2^-i x_i
if y > 0 then σ_i+1 := +1
         else σ_i+1 := -1
endif
endwhile.
```

Figure 12. CORDIC Rotation Algorithm.

A systolic array has simple and regular communication and control structures, and this is a substantial advantage over other designs and implementations. Additionally, we want to allow the computations to proceed concurrently with the input, in order to maximize the throughput. This is known as pipelining [Ref. 3].

Figure 13 shows a typical system. As said before, the array is simply a network of processors that are regularly connected. The data is continuously "pumped" through this structure, thereby minimizing overall execution time, since all the processors work in parallel.

It was shown that two processors would be necessary to solve Equation (3.1). Previously used configurations have consisted of a triangular array (as in Figure 13) to compute the upper triangular matrix, and a linear array to solve the system of equations.

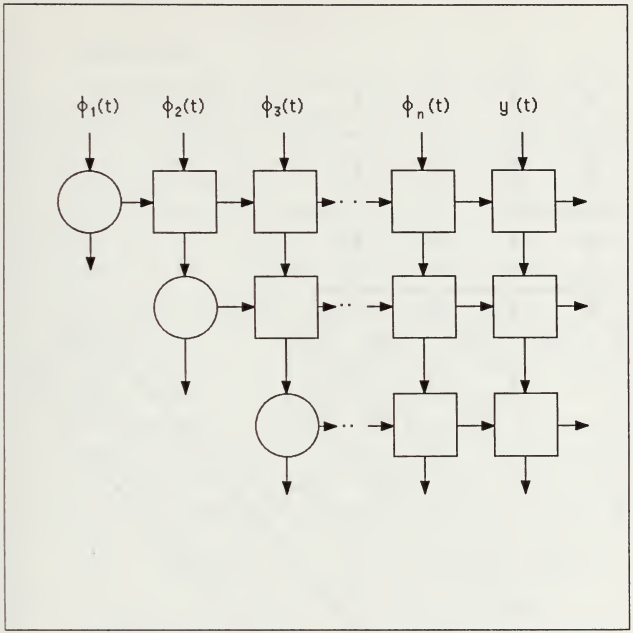


Figure 13. Systolic Array.

The entire systolic array is controlled by a single clock. Figure 14 shows the typical design for the case d (dimension) = 3, where d shows the number of columns which enter into the system. In this thesis we will use the alternative configuration in which the linear section is replaced by a second triangular section identical to the first one. This new design will be discussed later in this chapter. Both designs use a single clock signal to control operations. Before continuing on to discuss the alternative design, we will review the structures of the triangular and linear sections.

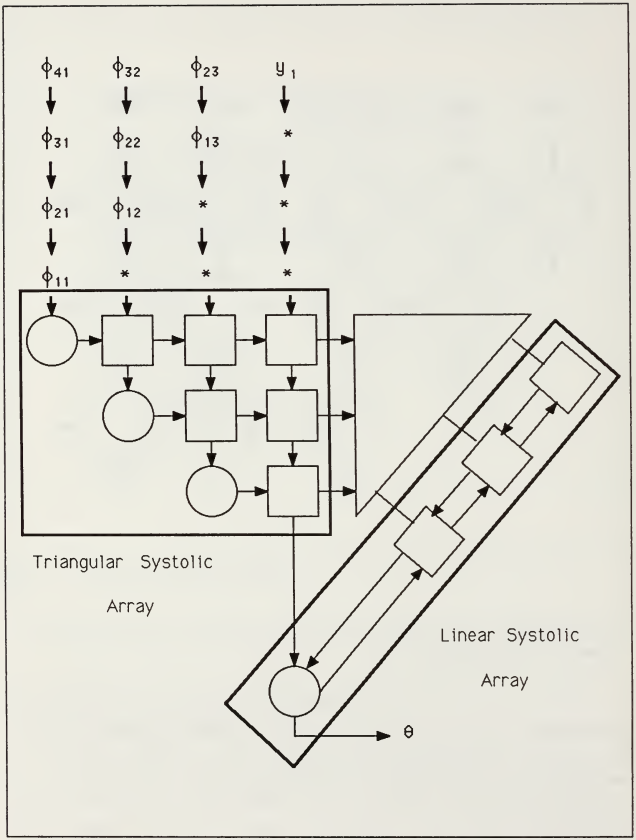


Figure 14. Systolic Array Procedure for Old Design.

1. Triangular Array

The triangular systolic array performs a sequence of Givens rotations. The CORDIC technique is a possible implementation for the rotations. The processors work simultaneously at each clock pulse. The data regression vector $\varphi(t)$ and output signal y(t) are inputs to the top of the array, and rotations are calculated at each clock cycle.

The triangular array consists of two types of cells: edge cells (or boundary cells) and internal cells. The edge cells are represented by the circles and the boundary cells are represented by the squares as seen in the Figure 13 and Figure 14. Figure 9 defines the operations of these cells. The edge cell computes the rotation parameters c and s as seen in Equation (3.2). Each cell of the triangular array stores an element of the upper triangular matrix R(n) from Equation (3.1), and it is initialized to zero for internal cells and to $\sigma_0 I$ for the edge cells as mentioned before. The function of each row of processing cells in the triangular systolic array section is to combine one row of the stored triangular matrix with a vector of data received from the above cells in such a way that the leading element of the received data vector is annihilated. The data vector so obtained is then passed downward on to the next row of cells. The boundary cells in each row of the section computes the rotation parameters and then passes them to the right on the next clock cycle. The internal cells apply the same rotation parameter to all other elements in the received data vector.

A delay of one clock cycle per cell is incurred when passing the rotation parameters along a row. That is why it is necessary to "skew" the input data as seen in Figure 14, so that the input data interacts properly with the previously stored triangular matrix. Because the cells are operating simultaneously, the data in the system at any time t consists of values from (2n) different matrices. Figure 15 demonstrates this for a system with n = 3. In this figure, we can see that at time (t + 5), there are also values present from the five previous matrices (i.e., t + 4,t + 3,.....,t). In order to get all the cells in the array to a similar time state, the array would have to be clocked an additional 2n-1 (five) cycles, feeding zeros as input where necessary. At the completion, all cells will be at the same time (t + 5) [Ref. 7].

Note that at the same time the triangularization process is being carried out, the column vector β_{n1} is also being computed by the rightmost column of internal cells using y(n) as its input.

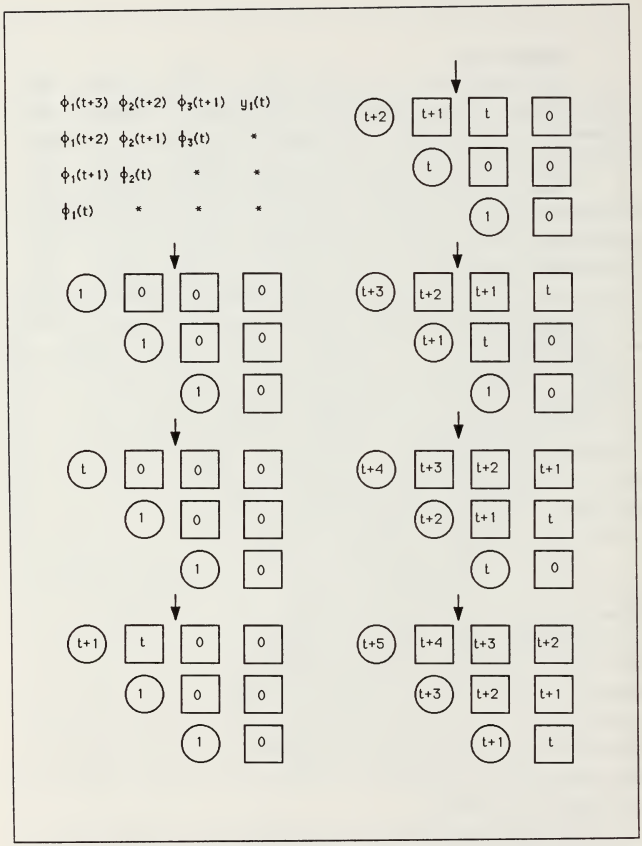


Figure 15. Data Flow in Triangular Section.

At the end of the triangularization period (N), we are ready to solve for $\theta_{(d-1),N}$. The data in this triangular array is clocked out to the next array section that will compute the parameters.

2. Linear Array

The linear systolic array has been used in previous implementations to solve for the estimated parameters. The linear section consists of one boundary cell and (d-1) internal cells as seen in Figure 14.

The operation of the cells as they compute the parameters are shown in Figure 16. Note that the cells are different from those in the triangular array, increasing to four the number of unique cells necessary in the combined system.

It is shown in [Ref. 3] that the time required to solve for θ using the linear array is equal to 2d. At the end of the period, the parameters are used as initial values for the triangular array, and the triangular section again begins the operation.

We now replace the linear section with a second triangular section, and discuss the differences between these two design approaches.

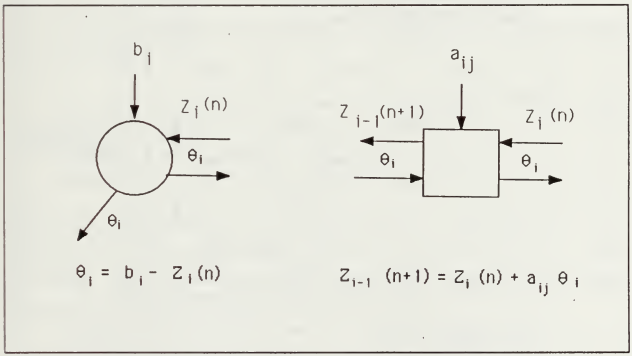


Figure 16. Definition of Cell Operation for Linear Section.

3. The Use of a Second Triangular Array as the Solution Section

An alternative implementation can be obtained by solving θ as shown in Figure 17.

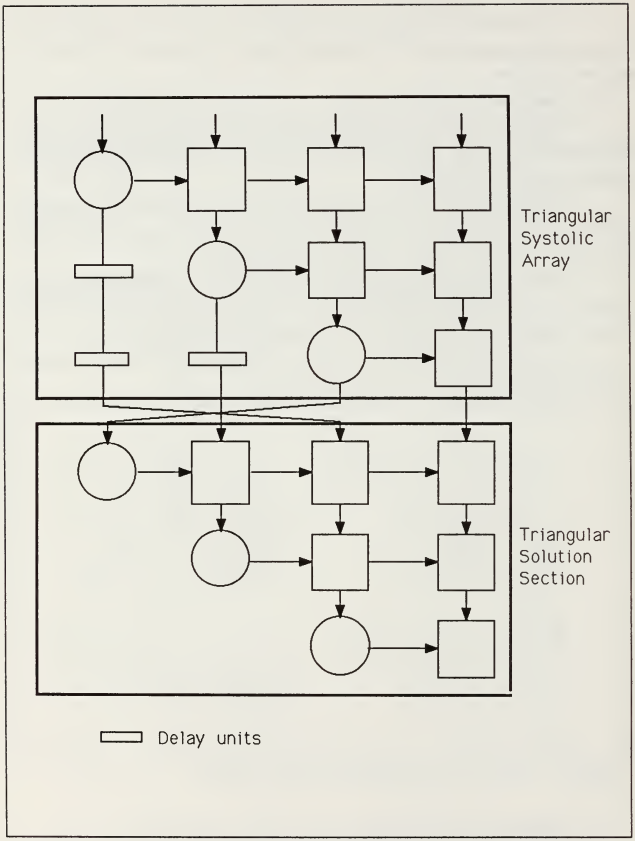


Figure 17. Systolic Array Procedure for New Design.

In this implementation, at the end of the triangularization period, the data is passed from the first triangular section to the second triangular section in a reversed position. The second triangular array performs the same type of operations as the first, therefore, the cells are identical.

The reason of using another triangular section is that, by proper combinations of rows, we can force zeros into all elements of a given row but one, so that we can solve for each of the parameters. Also, the fact that orthogonal operations are used makes it more robust in the presence of numerical errors [Ref. 7]. To see how this works, consider an arbitrary set of equations in upper triangular form as in Equation (3.14). Now x_3 is simply solved as b_3/a_{33}. To solve for x_2, we can force a_{23} to zero by a linear combination of rows two and three. Then x_2 is found to be b_2/a_{22}. Similarly, by row operations on row 1,2 and 3 we can make $a_{12} = a_{13} = 0$ in row one, and x_1 is found to be b_1/a_{11}. This type of operations are exactly what the triangular array was designed to do.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{vmatrix} \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix} = \begin{vmatrix} b_1 \\ b_2 \\ b_3 \end{vmatrix} \tag{3.14}$$

Figure 18 shows these operations in matrix format, and as mentioned before data is in reversed position. Notice that the array is initialized to all zeros here, whereas the first triangular section is initialized to $\sigma_0\theta_{x,y}$ and $\sigma_0 I$.

To understand how the system operates, recall the first triangular array at time N. Now we must feed the data down into the second triangular section in an appropriate manner so that we can solve for the parameters. The same delay (one clock cycle per cell) in propagation of data applies to this triangular section as it does in the first section. That is why we must carefully choose when to sample the array in order to get the correct values with which to calculate the solution.

Figure 19 illustrates the data flow for a simple system where d = 3. The input data is skewed as it was in the triangular section. It can be seen that the values a_{33}, a_{22} , and a_{11} are available at times N+1, N+4, and N+7 respectively. Similarly, the values of b_3, b_2, and b_1 are available at times N+4, N+6, and N+8. In general, for any size system n, the coefficients a_{ii} and outputs b_i are available as shown in Table 2. Note from the figure that the coefficients are "picked off" from the edge cells at the appropriate times, while the outputs are found in the rightmost set of internal cells [Ref. 3].

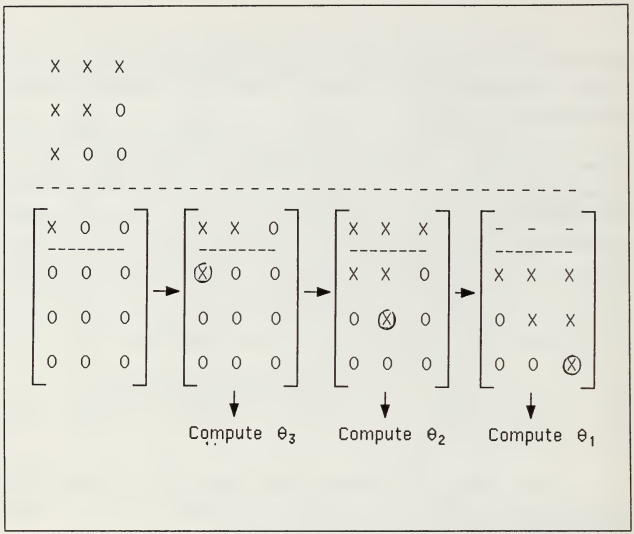


Figure 18. Solution of System of Equations.

This new design operates slower than the linear system seen in [Ref. 7]. The time to solve for θ in this design is equal to the time until b_1 appears, which is equal to (3n - 1). This compares to 2n (or 2d) in previous implementations; hence, n-1 more clock cycles are required.

On the other hand, simplification is gained in the manufacturing process since the number of types of cells is reduced. The tradeoffs to be considered are simplicity (cost) versus speed.

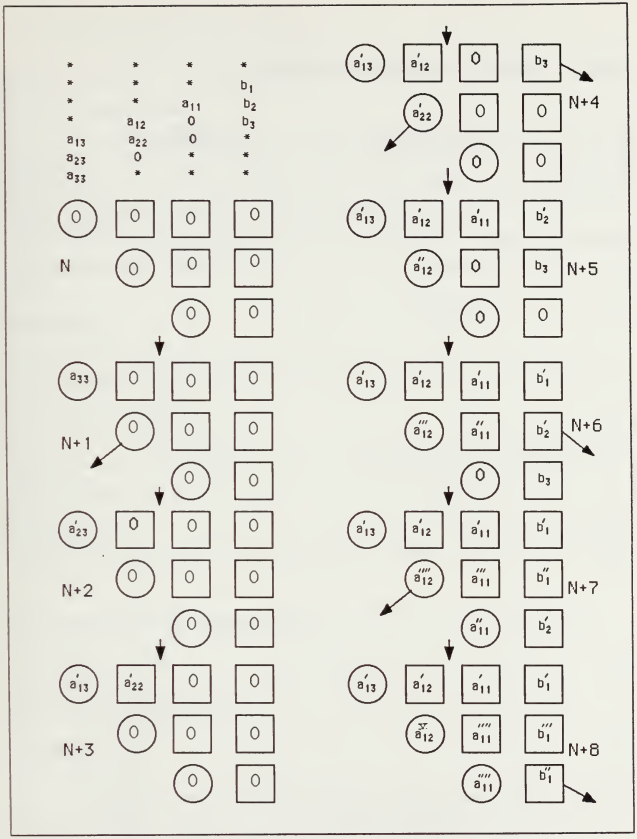


Figure 19. Data Flow in the Second Triangular Array.

Table 2. TIMES OF AVAILABILTY OF DATA.

Coefficient	Times
$a_{n,n}$	N+1
$a_{n-1,n-1}$	N+4
$a_{n-2,n-2}$	N+7
$a_{n-3,n-3}$	N+10
⋮	⋮
b_n	N + (n + 1)
b_{n-1}	N + (n + 3)
b_{n-2}	N + (n + 5)
b_{n-3}	N + (n + 7)

IV. SIMULATION STUDY

1. Model Equation

In general, a linear model can be expressed by either the difference equation relating the input and output sequence, or in a regression form as mentioned in the previous Chapters. In the example below, we will consider the problem of estimating the unknown parameters both with and without noise present.

In our simulation program, we will use the first order model which has the discrete time transfer function such that

$$H(z) = \frac{b_1}{z - a_1} \tag{4.1}$$

which corresponds to the linear difference equation

$$y(t) + a_1 y(t-1) = b_1 u(t-1) \tag{4.2}$$

where u(t) and y(t) are the input and output sequences respectively. Equation (4.2) can be expressed as

$$\theta^T = [a_1, b_1] \tag{4.3}$$

$$\varphi^T(t) = [y(t-1), u(t-1)] \tag{4.4}$$

This corresponds to the regression model

$$y(t) = \theta^T \varphi(t) \tag{4.5}$$

If the model has noise we obtain

$$y(t) = \theta^T \varphi(t) + v(t) \tag{4.6}$$

For proper system identification the input sequence must contain a sufficient number of frequency components. In our simulating program we used the sine wave

$$u(t) = \sin(\frac{\pi t}{10}) + \sin(\frac{2\pi t}{10}) + \sin(\frac{5\pi t}{10}) + \sin(\frac{6\pi t}{10}) \tag{4.7}$$

For identification purposes the input must be sufficently rich in frequencies to excite all modes of the system. If we have fourth order system we should have at least four different sinusoidal components.

In our simulation model we chose the values of the parameters as $a_1 = 0.5$ and $b_1 = 2$. In the parameter estimation problem, these values will be assumed to be unknown.

2. Noise Model

The noise term v(t) is a sequence of independent random variables with zero mean values. The noise is added to both the incoming signal u(t) and measured output y(t).

3. Choice of Initial Values

For a recursive algorithm, we need to initialize the systolic array with an initial parameter estimate $\sigma_0\theta(0)$ and $\sigma_0 I$ in Equation (2.28). Here σ_0 is related to the confidence of initial condition of $\theta(0)$. If some prior information about $\theta(0)$ is available, it should be used for determining proper values of $\theta(0)$. If no prior information is available we choose $\theta(0) = 0$.

4. Choice of Block Length

In our simulation program we have used four different block lengths both without noise and with noise in order to test the convergence rates in the different cases. We chose N = 3, 7, 10, 15 values for the block lengths, where N identifies the block sizes.

5. Floating Point and Fixed Point Operations

During simulation, the use of floating point versus fixed point arithmetic is considered. Fixed point arithmetic operations are performed using simple shift operations and finite registers. Because of this, they are simpler to implement than floating point operations. On the other hand, since input and output data values do not naturally appear as integer values, there is some concern over loss of accuracy. The solution to the latter problem is to scale all the numbers so that they stay within the limits of the fixed registers.

A. NOISELESS MEASUREMENTS

This is an ideal case. Figure 20 through 27 illustrate the results of these simulations. In these figures the estimated parameters (θ_1 and θ_2) are plotted along the vertical axis, the block number is plotted along the horizontal axis. In the following we will discuss the results for floating point and fixed point arithmetic.

1. Results Using Floating Point Arithmetic

Figures 20 through 23 illustrate the floating point results for block lengths of N = 3, 7, 10 and 15 respectively. As seen from the figures the parameters exhibit the fastest rate of converge when N = 7. Even though the estimated parameter a_1 converges very fast at N = 15, the other estimated parameter b_1 converges slowly in this block length. As it can seen from the Table 3, the case of N = 7 requires minimum number of clock cycles, therefore in this case we should choose a block length of seven.

2. Results Using Fixed Point Arithmetic

This situation has been simulated by adding the random noise to the measurements and the computations, so to account for round-off errors.

Figure 24 through 27 illustrate the fixed point results for the same conditions. Notice that the parameters converge as fast as floating point operations. As seen from Table 4, for this case again, N = 7 requires minimum number of clock cycles. This indicates that the degradation due to the implementation is not dramatic for the fixed point processors. When we consider the simplicity of fixed point processors, this is a distinct advantage.

Table 3. THE RATE OF CONVERGE FOR FLOATING POINT(NO NOISE).

BLOCK LENGTH(N)	BLOCK NUMBER AT CONVERGE	TOTAL CLOCK CYCLES
3	30	90
7	12	84
10	14	140
15	11	165

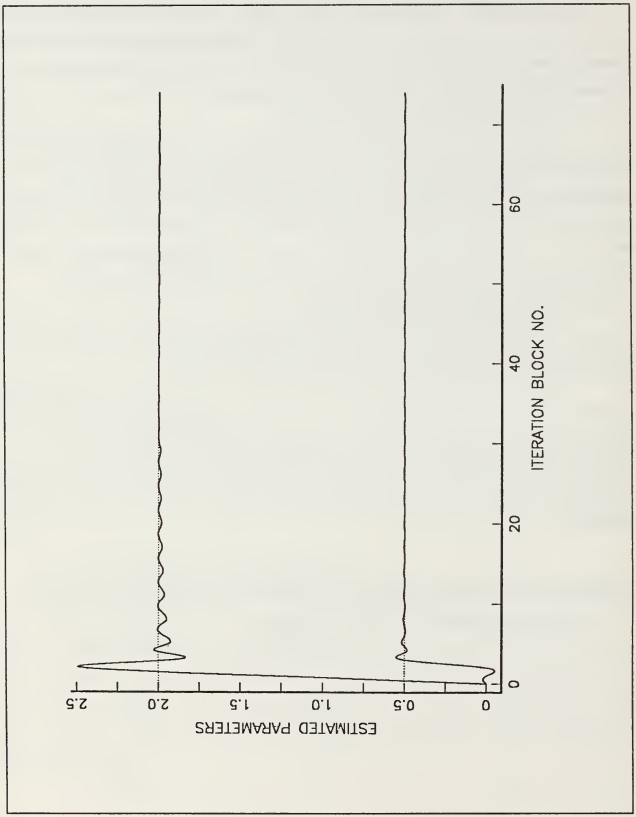


Figure 20. Floating Point Operation for N = 3 (No Noise).

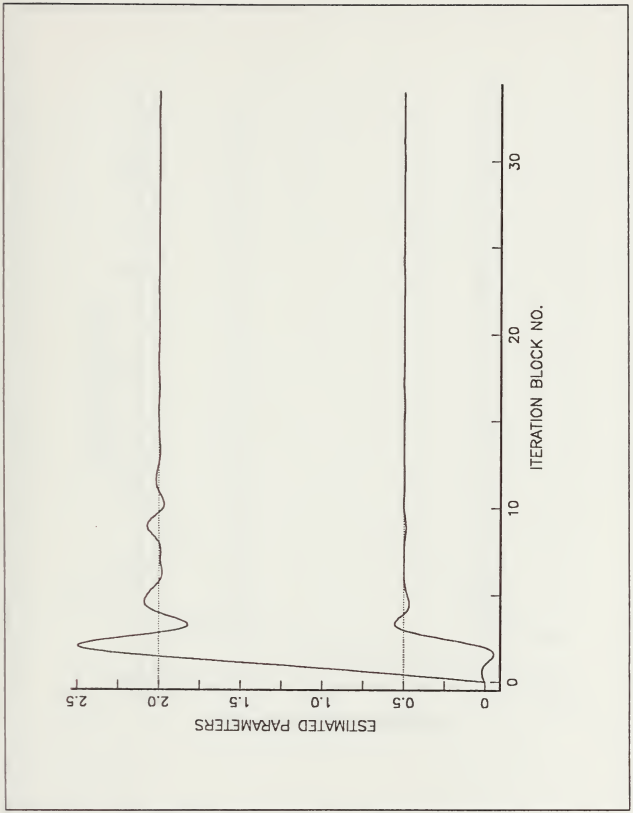


Figure 21. Floating Point Operation for N = 7 (No Noise).

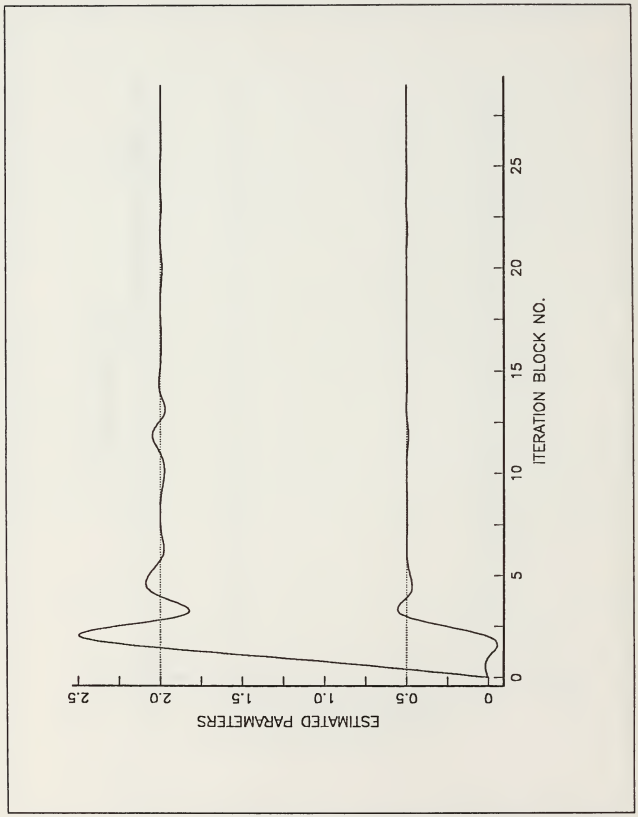


Figure 22. Floating Point Operation for N = 10 (No Noise).

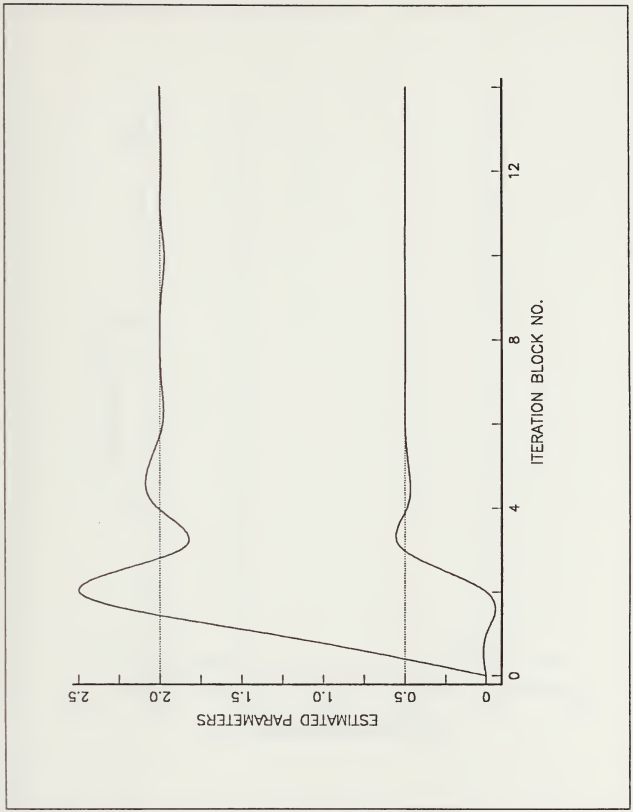


Figure 23. Floating Point Operation for N = 15 (No Noise).

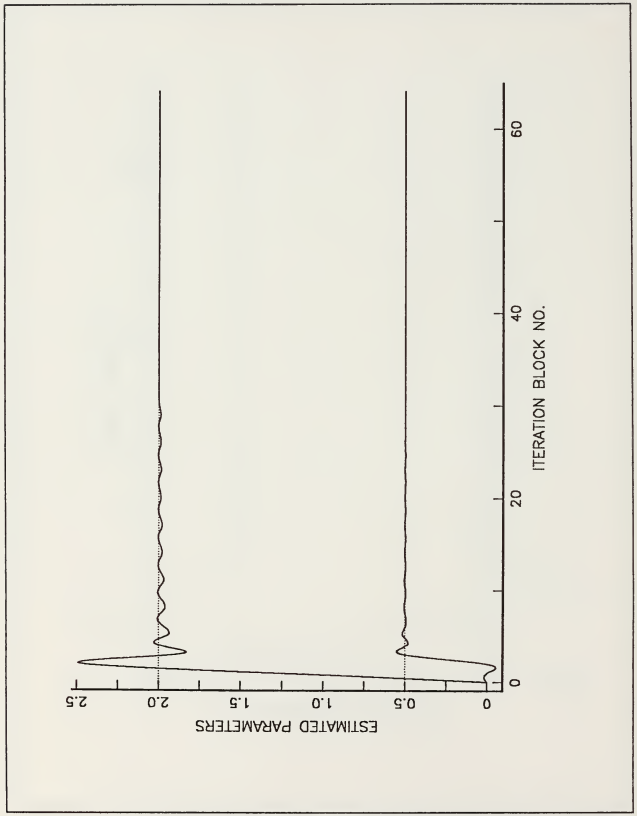


Figure 24. Fixed Point Operation for N = 3 (No Noise).

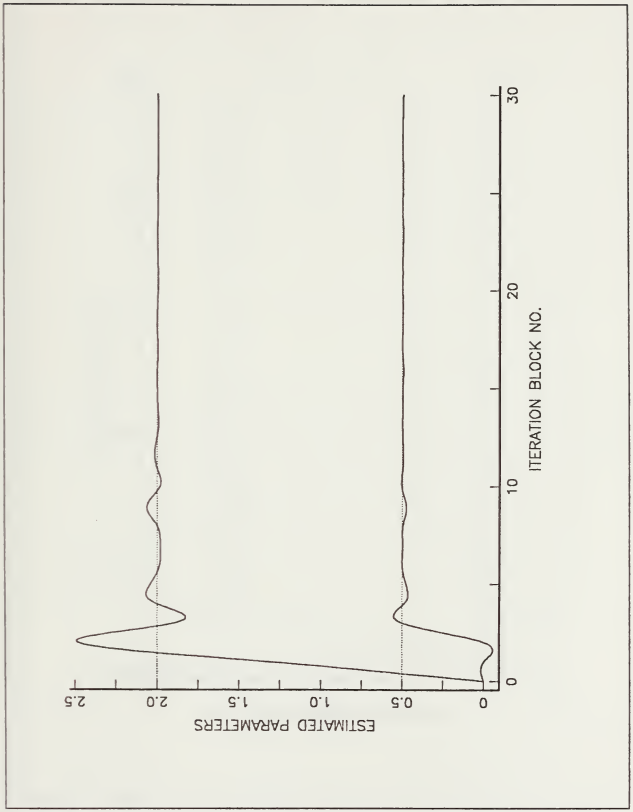


Figure 25. Fixed Point Operation for N = 7 (No Noise).

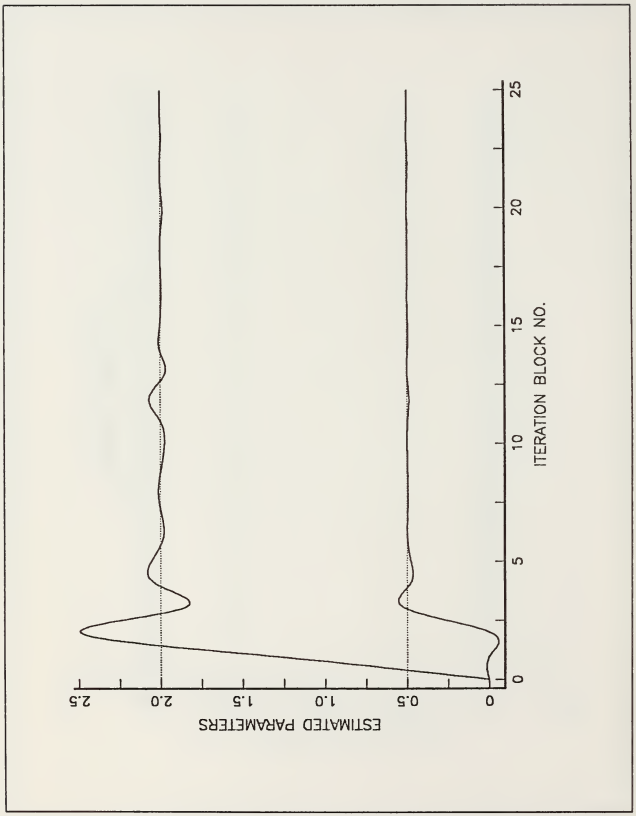


Figure 26. Fixed Point Operation for N = 10 (No Noise).

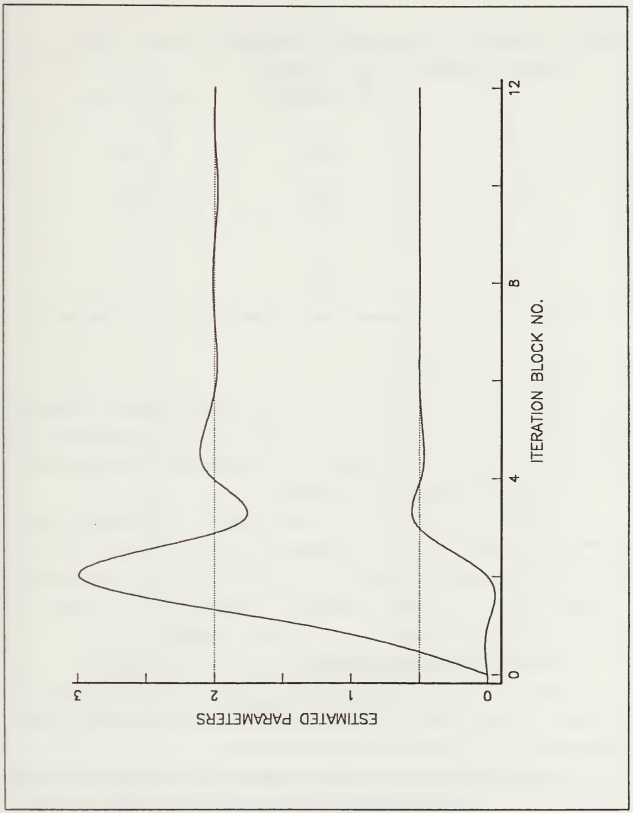


Figure 27. Fixed Point Operation for N = 15 (No Noise).

Table 4. THE RATE OF CONVERGE FOR FIXED POINT(NO NOISE).

BLOCK LENGTH(N)	BLOCK NUMBER AT CONVERGE	TOTAL CLOCK CYCLES
3	30	90
7	12.5	87.5
10	15	150
15	10.5	157.5

B. NOISY MEASUREMENTS

As mentioned before the noise term v(t) is a sequence of independent random variables with zero mean. We used a variance of 0.5 for these random variables. As expected in a real system, the noise is added to both incoming signal u(t) and output y(t).

1. Results Using Floating Point Arithmetic

Figures 28 through 31 show the results for the four different block lengths. As seen from the figures, for $N=3$ the parameters are most affected. For $N=7$ or $N=10$, the parameters are less affected by the noise. When the block length is equal to 7, the parameters have converged reasonably well within eight blocks (56 cycles). For $N=15$, we see that the parameters are least affected by the noise.

2. Results Using Fixed Point Arithmetic

Figures 32 through 35 show the results for fixed point implementation. Again we see that they exhibit similar performance as in the floating point cases. Effects of block length are almost the same as described previously.

To simulate fixed point behavior we added random disturbances in the computations within the cells. In particular, the factors c and s are affected by round-off errors which we can simulate in this fashion. A similar approach has been taken in [Ref. 9] in the analysis of a systolic array implementation of the projection operator.

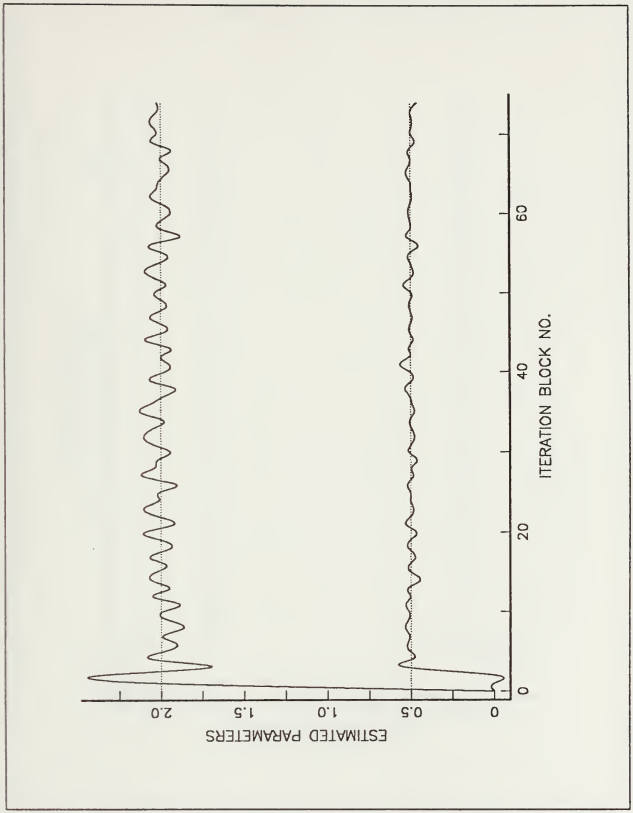


Figure 28. Floating Point Operation for N = 3 (With Noise).

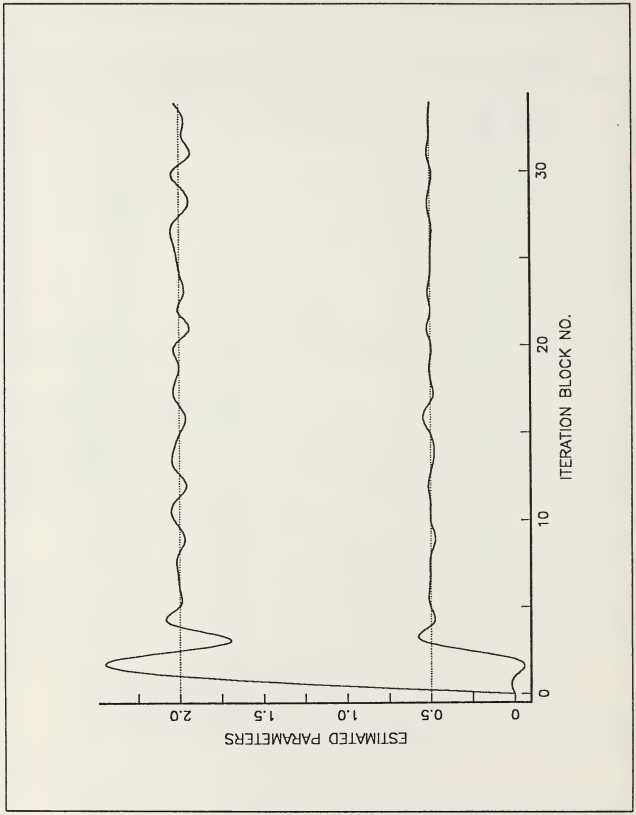


Figure 29. Floating Point Operation for N = 7 (With Noise).

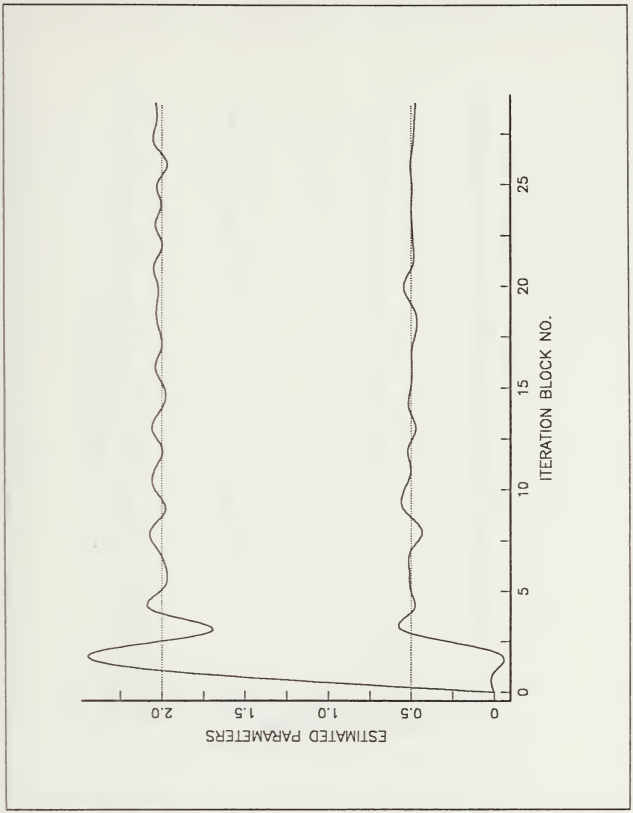


Figure 30. Floating Point Operation for N = 10 (With Noise).

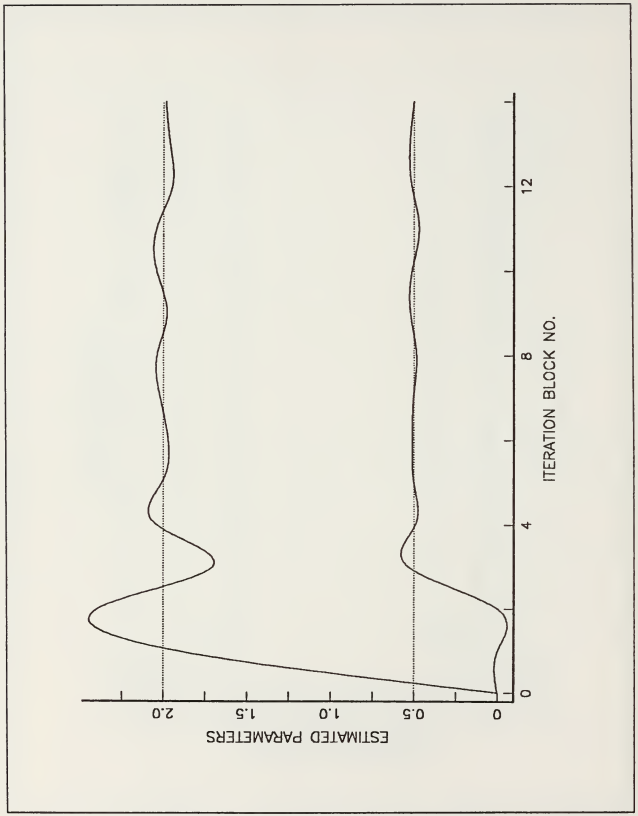


Figure 31. Floating Point Operation for N = 15 (With Noise).

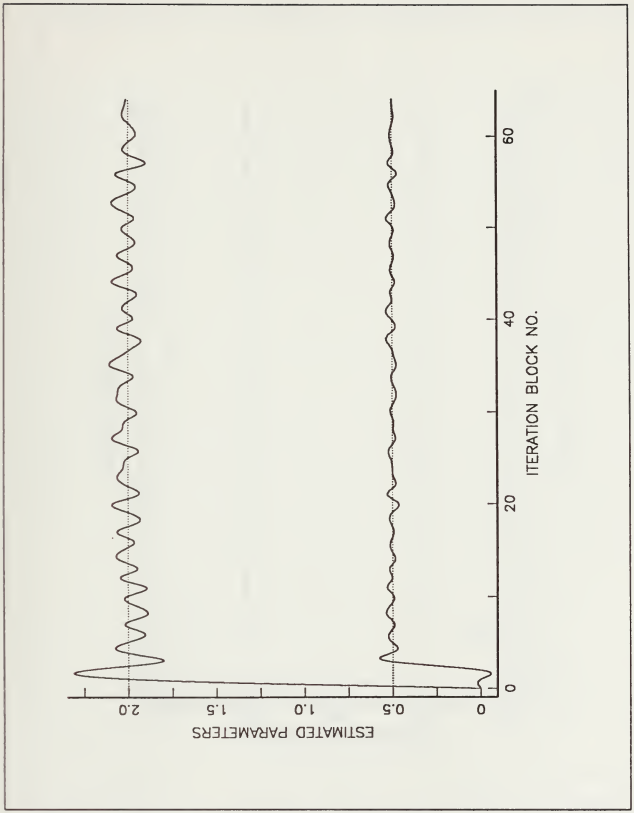


Figure 32. Fixed Point Operation for N = 3 (With Noise).

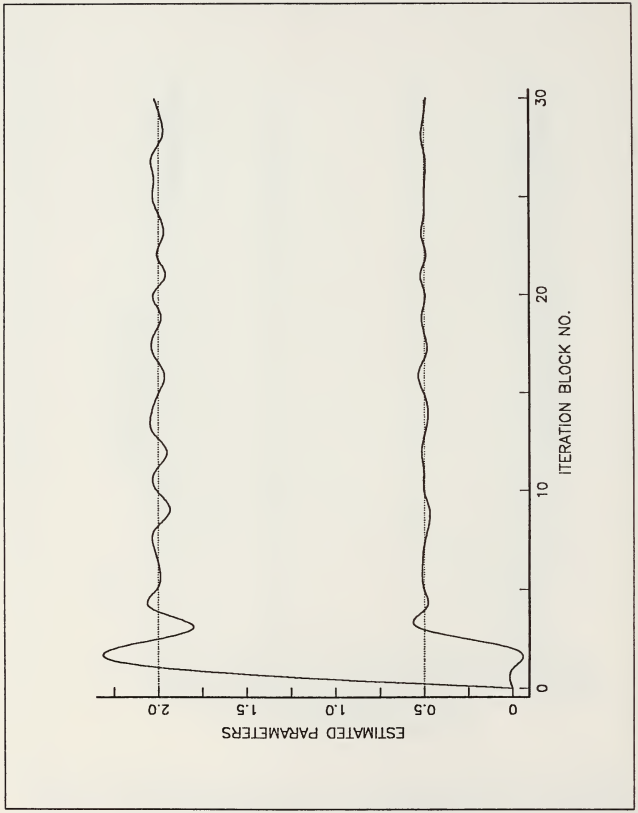


Figure 33. Fixed Point Operation for N = 7 (With Noise).

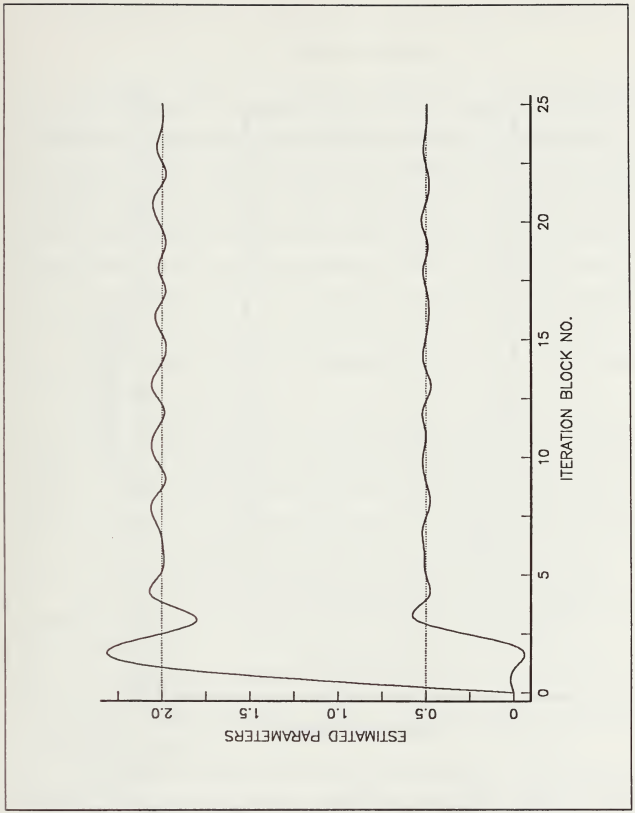


Figure 34. Fixed Point Operation for N = 10 (With Noise).

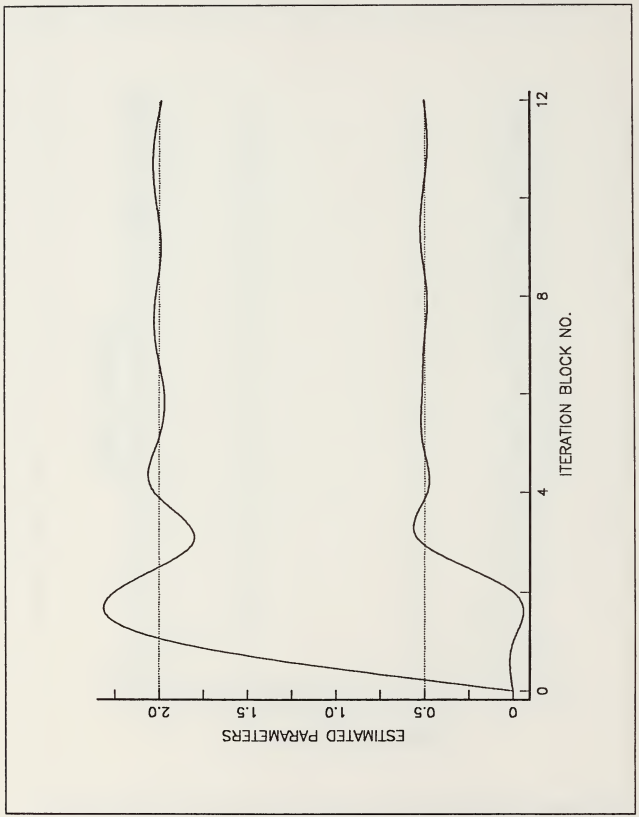


Figure 35. Fixed Point Operation for N = 15 (With Noise).

V. CONCLUSIONS

In the previous chapters we explained the estimation of the model using a parallel structure and recursive least squares algorithm. In this chapter we will discuss the tradeoffs, advantages and disadvantages of the systems that we have investigated.

The recursive algorithm, based on recursive least squares with covariance resetting, is redesigned in order to make it suitable for implementation on a VLSI chip and also modify the systolic array in order to reduce computing time and complexity. The difference in adaptive control is that the estimator has to operate recursively on subsequent blocks of data, so that the estimated parameters converge to the respective correct values. This convergence is guaranteed by initializing each estimation with the parameter values from the previous one, and by persistency of excitation of the external inputs.

We simulated several possible conditions to see the response of the parallel algorithm. Almost in each condition the parameters converged close to their own values, even in the presence of noise.

This parallel algorithm is different from other similar algorithms in the fact that we replaced two different processors by two identical ones. There was a total of four different computing cells (two for triangular section, two for linear section) in the old design. For the new design we need only two different cells for the whole system as explained in Chapter III. The new design requires more total cells than the old design. For a fourth order system (d=4), new design requires 10 additional cells from the formula of $\frac{1}{2}d(d+1)$. However, additional cells are not expensive in a VLSI schema.

As described in Chapter III additional (d-1) clock cycles are required to operate the second triangular section. That is why a second triangular section is a little bit slower than the linear section.

In this thesis we also compared floating point operations with fixed point operations. For floating point operations we saw that the convergence rate increases with the block length. Also when the block length is small, the identification is more sensetive to the presence of noise.

We got almost the same results for fixed point operations. If we consider the simplicity of the fixed point processor, a significant advantage to use the fixed point arithmetic is due to its simplicity.

APPENDIX A. COMPUTER PROGRAM I

A. PURPOSE OF THE PROGRAM

This program converts a given data matrix to an upper triangular matrix by using Givens rotation algorithm. There are two classes of cells. Therefore, two different types of subroutines are used. As explained in Chapter III this upper triangular matrix again convert to another upper triangular matrix and meanwhile the unknown parameters are computed.

All elements of the matrix are given interactively. This program can solve 49 × 50 matrix. If run this program, should be extended virtual storage capacity to 1500K in advance due to the large array.

```
C      **********VARIABLE DECLARATION**********
       IMPLICIT REAL*8 (A-H,O-Z)
       REAL*8 A,C,S,U,LU,AOUT,AA,ED,IN,AAA,CC,SS,UU,AAOUT,LLU,EED,
      +IIN,THETA
       DIMENSION A(50,50),C(0:50,0:50),S(0:50,0:50),U(0:50,0:50),
      +LU(0:50,0:50),AOUT(0:50,0:50),AA(50,50),ED(0:50,0:50),
      +IN(0:50,0:50),AAA(50,50),CC(0:50,0:50),SS(0:50,0:50),
      +UU(0:50,0:50),LLU(0:50,0:50),AAOUT(0:50,0:50),
      +EED(0:50,0:50),IIN(0:50,0:50),THETA(50)
       INTEGER I,J,K,M,N
C      ************************************************
C      **********VARIABLE DEFINIATION*********
C      A(I,J)   = DATA MATRIX
C      ED(I,J)  = FIRST TRIANGULAR MATRIX
C      EED(I,J) = SECOND TRIANGULAR MATRIX
C      THETA(K)     = UNKNOWN PARAMETERS
C      ************************************************
C      **************DATA ENTRANCE***************
       PRINT *,'ENTER THE NUMBER OF COLUMNS M='
       READ(5,*)M
       PRINT *,'ENTER THE NUMBER OF ROWS N='
       READ(5,*)N
       PRINT *,' '
       DO 1 I=1,N
         DO 2 J=1,M
           WRITE(6,3)I,J
3          FORMAT('ENTER A(',I2,I2,')')
           READ(5,*) A(I,J)
2        CONTINUE
1      CONTINUE
       PRINT *,'DATA MATRIX'
       DO 4 I=1,N
```

```
        WRITE(6,*)(A(I,J),J=1,M)
4     CONTINUE
      PRINT *,' '
      DO 5 I=1,N
        DO 6 J=1,M
          AA(I,J)=A(N-(I-1),J)
6       CONTINUE
5     CONTINUE
      DO 7 I=1,2,3
        DO 8 J=1,M
          AA(N+I,J)=0.0
8       CONTINUE
7     CONTINUE
      DO 9 I=1,2,3
        DO 10 J=1,M
          AOUT(I,J)=0.0
10      CONTINUE
9     CONTINUE
      DO 11 I=1,N+1
        DO 12 J=1,M
          U(I,J)=AA(I,J)
12      CONTINUE
11    CONTINUE
C     ************************************************
C     **********FIRST TRIANGULARIZATION********
      DO 13 K=1,N
        DO 14 J = K,M
          DO 15 I = 1,N-K+2
          IF (J.EQ.K)THEN
          CALL EDGE(U(I,J),AOUT(I,J),AOUT(I+1,J),C(I,J),S(I,J))
          ED(I,J)=AOUT(I,J)
          ELSE
          CALL INTERNAL(U(I,J),AOUT(I,J),C(I,J-1),S(I,J-1),AOUT(I+1,J),
     +LU(I,J),C(I,J),S(I,J))
          U(I-1,J)=LU(I,J)
          IN(I,J)=AOUT(I,J)
          ENDIF
15        CONTINUE
14      CONTINUE
13    CONTINUE
      DO 20 I=1,N
        DO 21 J=1,N+1-I
          ED(I,J)=0.0
21      CONTINUE
20    CONTINUE
      DO 22 J=M,2,-1
        DO 23 K=N+3-J,N+1
          DO 24 I=K,N+1
            ED(I,J)=IN(I,J)
24        CONTINUE
23      CONTINUE
22    CONTINUE
      PRINT *,'FIRST TRIANGULAR MATRIX'
      DO 29 I=N+1,2,-1
        WRITE(6,*)(ED(I,J),J=1,M)
29    CONTINUE
```

```
      PRINT *,' '
C     **************DATA ENTRANCE***************
      DO 25 I=1,N
        DO 26 J=1,M-1
          AAA(I,J)=ED(I+1,M-J)
26      CONTINUE
25    CONTINUE
      DO 27 J=M,M+1,2
        DO 28 I=1,N
          AAA(I,J)=ED(I+1,M)
28      CONTINUE
27    CONTINUE
      DO 30 I=1,2,3
        DO 31 J=1,M
          AAA(N+I,J)=0.0
31       CONTINUE
30     CONTINUE
      DO 32 I=1,2,3
        DO 33 J=1,M
          AAOUT(I,J)=0.0
33      CONTINUE
32    CONTINUE
      DO 34 I=1,N+1
        DO 35 J=1,M
          UU(I,J)=AAA(I,J)
35      CONTINUE
34    CONTINUE
C     ************************************************
C     *********SECOND TRIANGULARIZATION*******
      DO 36 K=1,N
        DO 37 J = K,M
          DO 38 I = 1,N-K+2
          IF (J.EQ.K)THEN
          CALL EDGE(UU(I,J),AAOUT(I,J),AAOUT(I+1,J),CC(I,J),SS(I,J))
          EED(I,J)=AAOUT(I,J)
          ELSE
          CALL INTERNAL(UU(I,J),AAOUT(I,J),CC(I,J-1),SS(I,J-1),
     +AAOUT(I+1,J),LLU(I,J),CC(I,J),SS(I,J))
          UU(I-1,J)=LLU(I,J)
          IIN(I,J)=AAOUT(I,J)
          ENDIF
38        CONTINUE
37      CONTINUE
        THETA(K)=IIN(2,M)/EED(2,K)
36    CONTINUE
      DO 39 I=1,N
        DO 40 J=1,M-I
          EED(I,J)=0.0
40      CONTINUE
39    CONTINUE
      DO 41 J=M,2,-1
        DO 42 K=M+2-J,N+1
          DO 43 I=K,N+1
            EED(I,J)=IIN(I,J)
43        CONTINUE
```

```
42      CONTINUE
41    CONTINUE
      PRINT *,'SECOND TRIANGULAR MATRIX'
      DO 44 I=N+1,2,-1
        WRITE(6,*)(EED(I,J),J=1,M)
44    CONTINUE
      PRINT *,' '
      PRINT *,'UNKNOWN PARAMETERS'
      WRITE(*,46)
46    FORMAT('',11X,'K',13X,'THETA(K)')
      DO 45 K=N,1,-1
        WRITE(6,*) N-K+1,THETA(K)
45    CONTINUE
      STOP
      END
C     ********************************************


C     ********************************************
C     **********SUBROUTINE  GROUPS FOR**********
C     **************CELLS FUNCTION***************
C     ********************************************


C     **************BOUNDARY CELL*****************
      SUBROUTINE EDGE (U,AIN,AOUT,C,S)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 U,AIN,AOUT,C,S
      IF(U.EQ.0.AND.AIN.EQ.0)THEN
      C=1.0
      S=0.0
      AOUT=U
      ELSE
      C=U/(SQRT(U**2.0D0+AIN**2.0D0))
      S=AIN/(SQRT(U**2.0D0+AIN**2.0D0))
      AOUT=SQRT(U**2.0D0+AIN**2.0D0)
      ENDIF
      RETURN
      END
C     ********************************************


C     **************INTERNAL CELL*****************
      SUBROUTINE INTERNAL (PU,AIN,PC,PS,AOUT,LU,LC,LS)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 PS,PC,PU,S,C,U,AOUT,LU,AIN,LC,LS
      LU=(-PU*PS)+(AIN*PC)
      AOUT=(PU*PC)+(AIN*PS)
      LC=PC
      LS=PS
      RETURN
      END
C     ********************************************
```

```
C     ********************************************
C     ********************************************
C     ******THE RESULTS OF THIS PROGRAM*******
ENTER THE NUMBER OF COLUMNS M=
3
ENTER THE NUMBER OF ROWS N=
2

ENTER A( 1 1)
1
ENTER A( 1 2)
2
ENTER A( 1 3)
3
ENTER A( 2 1)
4
ENTER A( 2 2)
5
ENTER A( 2 3)
6
    DATA MATRIX
    1.0000        2.0000        3.0000
    4.0000        5.0000        6.0000

    FIRST TRIANGULAR MATRIX
    4.1231        5.3357        6.5484
    0.0000        0.7276        1.4552

    SECOND TRIANGULAR MATRIX
    5.3851        4.0852        6.6850
    0.0000        0.5570       -0.5570

    UNKNOWN PARAMETERS
     K       THETA(K)
     1       -1.0000
     2        2.0000



ENTER THE NUMBER OF COLUMNS M=
4
ENTER THE NUMBER OF ROWS N=
3

ENTER A( 1 1)
3
ENTER A( 1 2)
7
ENTER A( 1 3)
11
ENTER A( 1 4)
20
ENTER A( 2 1)
5
ENTER A( 2 2)
9
```

```
ENTER A( 2 3)
16
ENTER A( 2 4)
21
ENTER A( 3 1)
6
ENTER A( 3 2)
8
ENTER A( 3 3)
10
ENTER A( 3 4)
22
    DATA MATRIX
    3.0000        7.0000       11.0000       20.0000
    5.0000        9.0000       16.0000       21.0000
    6.0000        8.0000       10.0000       22.0000

    FIRST TRIANGULAR MATRIX
    8.3666       13.6256       20.6774       35.4982
    0.0000        2.8884        6.6670        7.3792
    0.0000        0.0000        2.2345       -3.2276

    SECOND TRIANGULAR MATRIX
   21.8403       13.7818        7.9211       35.5305
    0.0000        2.0151        2.3979        7.6038
    0.0000        0.0000        1.2269       -2.1812

    UNKNOWN PARAMETERS
     K       THETA(K)
     1       -1.7777
     2        5.8888
     3       -1.4444


ENTER THE NUMBER OF COLUMNS M=
5
ENTER THE NUMBER OF ROWS N=
4

ENTER A( 1 1)
6
ENTER A( 1 2)
9
ENTER A( 1 3)
14
ENTER A( 1 4)
26
ENTER A( 1 5)
31
ENTER A( 2 1)
42
ENTER A( 2 2)
14
ENTER A( 2 3)
57
```

```
ENTER A( 2 4)
38
ENTER A( 2 5)
3
ENTER A( 3 1)
9
ENTER A( 3 2)
21
ENTER A( 3 3)
42
ENTER A( 3 4)
57
ENTER A( 3 5)
48
ENTER A( 4 1)
12
ENTER A( 4 2)
19
ENTER A( 4 3)
23
ENTER A( 4 4)
45
ENTER A( 4 5)
58
    DATA MATRIX
     6.0000       9.0000      14.0000      26.0000      31.0000
    42.0000      14.0000      57.0000      38.0000       3.0000
     9.0000      21.0000      42.0000      57.0000      48.0000
    12.0000      19.0000      23.0000      45.0000      58.0000

    FIRST TRIANGULAR MATRIX
    44.9999      23.5333      69.6000      62.3333      32.0000
     0.0000      22.9168      26.4032      58.9561      73.2183
     0.0000       0.0000      14.0252       4.5607     -14.6452
     0.0000       0.0000       0.0000       3.4541       6.2125

    SECOND TRIANGULAR MATRIX
    85.9883      69.3000      32.7718      32.6206      72.8703
     0.0000      30.5859      -0.9184      28.4896     -35.7980
     0.0000       0.0000       2.0397       7.9060       4.9136
     0.0000       0.0000       0.0000       9.3125       4.7191

    UNKNOWN PARAMETERS
     K       THETA(K)
     1        0.5067
     2        0.4447
     3       -1.6290
     4        1.7985
```

APPENDIX B. COMPUTER PROGRAM II

A. PURPOSE OF THE PROGRAM

This program computes the estimated weight vector of least squares for first order, second order or third order model which has the discrete time transfer function such that

$$H(Z) = \frac{b_1 Z^2}{Z^3 + a_1 Z^2 + a_2 Z + a_3}$$

where the all coefficients are given interactively. In this program, addition to the previous program is used another two subroutines to supply the gaussian noise which corresponds to parameter v(t). As seen in the plots this program computes the unknown parameters by converging. In order to run this program should be extended virtual storage capacity to 2M.

```
C     **********VARIABLE DECLARATION**********
      REAL C,S,U,LU,AOUT,ED,IN,X,Y,SIG,UIN,THETA,V,R,BM,AAA,CC,SS,UU,
     +LLU,AAOUT,EED,IIN
      DIMENSION C(0:50,0:50),S(0:50,0:50),U(0:50,0:50),LU(0:50,0:50),
     +AOUT(0:50,0:50),ED(0:50,0:50),IN(0:50,0:50),X(50),UIN(0:50),
     +AAA(50,50),CC(0:50,0:50),SS(0:50,0:50),UU(0:50,0:50),
     +LLU(0:50,0:50),AAOUT(0:50,0:50),EED(0:50,0:50),
     +IIN(0:50,0:50),Y(-3:50),THETA(0:50)
      INTEGER I,J,K,M,N,L,T
C     ***********VARIABLE DEFINIATION**********
C     ED(I,J)  = FIRST TRIANGULAR MATRIX
C     EED(I,J) = SECOND TRIANGULAR MATRIX
C     ************************************************
C
500   PRINT *,'ENTER THE SIZE OF MATRIX N BY N   **N=?'
      READ *, N
      PRINT *,' N = ',N
      PRINT *,' '
      PRINT *,'ENTER THE ORDERS OF DEN. OF DIFF. EQ.  **NO=?'
      READ *, NO
      PRINT *,' NO = ',NO
      PRINT *,' '
      PRINT *,'HOW MANY BLOCKS DO YOU WANT TO ITERATE  **NBN=?'
      READ *, NBN
      PRINT *,' NBN = ',NBN
      PRINT *,' '
      PRINT *,'ENTER THE VALUE OF SIGMA  **SIG=?'
      READ *,SIG
      PRINT *,'SIG = ',SIG
```

```
 PRINT *,' '
 IF (NO.EQ.1) THEN
 PRINT *,'FORMAT OF 1ST ORDER DIFF.EQ.** Y(T) = B1*U(T-1)+A1*Y(T-1)
+'
 PRINT *,'ENTER THE COEFF. OF A1 ?'
 READ *,A1
 PRINT *,'A1 = ',A1

 PRINT *,'ENTER THE COEFF. OF B1 ?'
 READ *,B1
 PRINT *,'B1 = ',B1
 WRITE(*,46)

 ELSEIF (NO.EQ.2) THEN
 PRINT *,'FORMAT OF 2ND ORDER DIFF.EQ.** Y(T) = B1*U(T-1)+A1*Y(T-1)
++A2*Y(T-2)'
 PRINT *,'ENTER THE COEFF. OF A1 ?'
 READ *,A1
 PRINT *,'A1 = ',A1

 PRINT *,'ENTER THE COEFF. OF A2 ?'
 READ *,A2
 PRINT *,'A2 = ',A2

 PRINT *,'ENTER THE COEFF. OF B1 ?'
 READ *,B1
 PRINT *,'B1 = ',B1
 WRITE(*,47)

 ELSEIF (NO.EQ.3) THEN
 PRINT *,'FORMAT OF 3RD ORDER DIFF.EQ.** Y(T) = B1*U(T-1)+A1*Y(T-1)
++A2*Y(T-2)+A3*Y(T-3)'
 PRINT *,'ENTER THE COEFF. OF A1 ?'
 READ *,A1
 PRINT *,'A1 = ',A1

 PRINT *,'ENTER THE COEFF. OF A2 ?'
 READ *,A2
 PRINT *,'A2 = ',A2

 PRINT *,'ENTER THE COEFF. OF A3 ?'
 READ *,A3
 PRINT *,'A3 = ',A3

 PRINT *,'ENTER THE COEFF. OF B1 ?'
 READ *,B1
 PRINT *,'B1 = ',B1
 WRITE(*,48)

 ELSE
 PRINT *,'ERROR, PLEASE TRY AGAIN'
 PRINT *,'CHOOSE NO=1,2 OR 3'
 GOTO 500
 ENDIF
 NB=15*NO
 Z=1
```

```
      PI=4*ATAN(Z)
      P=PI/10
      BM=0
      R=0.0707
      IX=1
C     ****************************************
C     **************DATA ENTRANCE***************
      DO 1 LF=2,NO+2
        DO 2 KF=1,NO+2
          IF((LF.GT.KF).AND.((LF-KF).EQ.1)) THEN
          U(LF,KF)=SIG
          ELSE
          U(LF,KF)=0.0
          ENDIF
2       CONTINUE
1     CONTINUE
      IF(NO.EQ.1)THEN
      Y(0)=0.0
      UIN(0)=0.0
      Y(1)=A1*Y(0)+B1*UIN(0)
      ELSEIF(NO.EQ.2)THEN
      Y(-1)=0.0
      Y(0)=0.0
      UIN(0)=0.0
      Y(1)=A2*Y(-1)+A1*Y(0)+B1*UIN(0)
      ELSE
      Y(-2)=0.0
      Y(-1)=0.0
      Y(0)=0.0
      UIN(0)=0.0
      Y(1)=A3*Y(-2)+A2*Y(-1)+A1*Y(0)+B1*UIN(0)
      ENDIF
      DO 3 IT=1,NBN
       DO 4 J=1,NB
        CALL GAUSS(IX,R,BM,V)
        IF(NO.EQ.1)THEN
        U(1,1)=Y(J-1)
        U(1,2)=UIN(J-1)
        U(1,3)=Y(J)
        UIN(J)=SIN(P*J)+SIN(P*2*J)+SIN(P*5*J)+SIN(P*6*J)
        Y(J+1)=A1*Y(J)+B1*UIN(J)+V
        ELSEIF(NO.EQ.2)THEN
        U(1,1)=Y(J-2)
        U(1,2)=Y(J-1)
        U(1,3)=UIN(J-1)
        U(1,4)=Y(J)
        UIN(J)=SIN(P*J)+SIN(P*2*J)+SIN(P*5*J)+SIN(P*6*J)
        Y(J+1)=A1*Y(J)+A2*Y(J-1)+B1*UIN(J)+V
        ELSE
        U(1,1)=Y(J-3)
        U(1,2)=Y(J-2)
        U(1,3)=Y(J-1)
        U(1,4)=UIN(J-1)
        U(1,5)=Y(J)
        UIN(J)=SIN(P*J)+SIN(P*2*J)+SIN(P*5*J)+SIN(P*6*J)
        Y(J+1)=A1*Y(J)+A2*Y(J-1)+A3*Y(J-2)+B1*UIN(J)+V
```

```
        ENDIF
      DO 5 II=1,2,3
        DO 6 JJ=1,N+1
          AOUT(II,JJ)=0.0
          U(N+II,JJ)=0.0
 6      CONTINUE
 5    CONTINUE
C     *********FIRST TRIANGULARIZATION*******
C     ********************************************
      DO 7 K=1,N
        DO 8 J1 = K,N+1
          DO 9 I = 1,N-K+2
          IF (J1.EQ.K)THEN
          CALL EDGE(U(I,J1),AOUT(I,J1),AOUT(I+1,J1),C(I,J1),S(I,J1))
          ED(I,J1)=AOUT(I,J1)
          ELSE
          CALL INTERNAL(U(I,J1),AOUT(I,J1),C(I,J1-1),S(I,J1-1),
     +AOUT(I+1,J1),LU(I,J1),C(I,J1),S(I,J1))
          U(I-1,J1)=LU(I,J1)
          IN(I,J1)=AOUT(I,J1)
          ENDIF
9         CONTINUE
8       CONTINUE
7     CONTINUE
      DO 10 I=1,N
        DO 11 J2=1,N+1-I
          ED(I,J2)=0.0
11      CONTINUE
10    CONTINUE
      DO 12 J3=N+1,2,-1
        DO 13 K=N+3-J3,N+1
          DO 14 I=K,N+1
            ED(I,J3)=IN(I,J3)
14        CONTINUE
13      CONTINUE
12    CONTINUE
      PRINT *,' '
C     **************DATA ENTRANCE**************
C     *******FOR SECOND TRIANGULAR ARRAY******
      DO 15 I=1,N
        DO 16 J4=1,N
          AAA(I,J4)=ED(I+1,N+1-J4)
16      CONTINUE
15    CONTINUE
      DO 17 J5=N+1,N+2,2
        DO 18 I=1,N
          AAA(I,J5)=ED(I+1,N+1)
18      CONTINUE
17    CONTINUE
      DO 20 I=1,2,3
        DO 21 J6=1,N+1
          AAA(N+I,J6)=0.0
21      CONTINUE
20    CONTINUE
      DO 22 I=1,2,3
        DO 23 J7=1,N+1
```

```
          AAOUT(I,J7)=0.0
23      CONTINUE
22    CONTINUE
      DO 24 I=1,N+1
        DO 25 J8=1,N+1
          UU(I,J8)=AAA(I,J8)
25      CONTINUE
24    CONTINUE
C     ****************************************
C     *********SECOND TRIANGULARIZATION*******
      DO 26 K=1,N
        DO 27 J9 = K,N+1
          DO 28 I = 1,N-K+2
          IF (J9.EQ.K)THEN
          CALL EDGE(UU(I,J9),AAOUT(I,J9),AAOUT(I+1,J9),CC(I,J9),SS(I,J9)
     +)
          EED(I,J9)=AAOUT(I,J9)
          ELSE
          CALL INTERNAL(UU(I,J9),AAOUT(I,J9),CC(I,J9-1),SS(I,J9-1),
     +AAOUT(I+1,J9),LLU(I,J9),CC(I,J9),SS(I,J9))
          UU(I-1,J9)=LLU(I,J9)
          IIN(I,J9)=AAOUT(I,J9)
          ENDIF
28        CONTINUE
27      CONTINUE
26    CONTINUE
      PRINT *,' '
C     ************************************************
      DO 29 I=1,N
        DO 30 J2=1,N+1-I
          EED(I,J2)=0.0
30      CONTINUE
29    CONTINUE
      DO 31 J3=N+1,2,-1
        DO 32 K=N+3-J3,N+1
          DO 33 I=K,N+1
            EED(I,J3)=IN(I,J3)
33        CONTINUE
32      CONTINUE
31    CONTINUE
      DO 34 MM=1,N+1
        DO 35 LL=2,N+1
         UU(LL,MM)=EED(N+3-LL,MM)
35      CONTINUE
34    CONTINUE
       IF(NO.EQ.1)THEN
       IF(UU(3,3).EQ.0.AND.UU(3,2).EQ.0)THEN
       THETA(2)=0.0
       ELSE
       THETA(2)=UU(3,3)/UU(3,2)
       THETA(1)=(UU(2,3)-(THETA(2)*UU(2,2)))/UU(2,1)
       WRITE(6,*) THETA(1),THETA(2)
       ENDIF
       ELSEIF(NO.EQ.2)THEN
       IF(UU(4,4).EQ.0.AND.UU(4,3).EQ.0)THEN
       THETA(3)=0.0
```

```
      ELSE
      THETA(3)=UU(4,4)/UU(4,3)
      THETA(2)=(UU(3,4)-(THETA(3)*UU(3,3)))/UU(3,2)
      THETA(1)=(UU(2,4)-(THETA(3)*UU(2,3)+THETA(2)*UU(2,2)))/UU(2,1)
      WRITE(6,*) THETA(1),THETA(2),THETA(3)
      ENDIF
      ELSEIF(NO.EQ.3)THEN
      IF(UU(5,5).EQ.0.AND.UU(5,4).EQ.0)THEN
      THETA(4)=0.0
      ELSE
      THETA(4)=UU(5,5)/UU(5,4)
      THETA(3)=(UU(4,5)-(THETA(4)*UU(4,4)))/UU(4,3)
      THETA(2)=(UU(3,5)-(THETA(4)*UU(3,4)+THETA(3)*UU(3,3)))/UU(3,2)
      THETA(1)=(UU(2,5)-(THETA(4)*UU(2,4)+THETA(3)*UU(2,3)+THETA(2)*
     +UU(2,2)))/UU(2,1)
      WRITE(6,*) THETA(1),THETA(2),THETA(3),THETA(4)
      ENDIF
      ENDIF
4     CONTINUE
      IF(NO.EQ.1)THEN
      Y(0)=Y(NB)
      UIN(0)=UIN(NB)
      Y(1)=Y(NB+1)
      ELSEIF(NO.EQ.2)THEN
      Y(-1)=Y(NB-1)
      Y(0)=Y(NB)
      UIN(0)=UIN(NB)
      Y(1)=Y(NB+1)
      ELSE
      Y(-2)=Y(NB-2)
      Y(-1)=Y(NB-1)
      Y(0)=Y(NB)
      UIN(0)=UIN(NB)
      Y(1)=Y(NB+1)
      ENDIF
3     CONTINUE
46    FORMAT('',17X,'THETA1',17X,'THETA2')
47    FORMAT('',17X,'THETA1',17X,'THETA2',18X,'THETA3')
48    FORMAT('',17X,'THETA1',17X,'THETA2',18X,'THETA3',19X,'THETA4')
      STOP
      END
C     ***********************************************
C     *********SUBROUTINE  GROUPS FOR*********
C     ************CELLS FUNCTION**************
C
C     ************BOUNDARY CELL**************
      SUBROUTINE EDGE (U,AIN,AOUT,C,S)
      REAL U,AIN,AOUT,C,S
      IF(U.EQ.0.AND.AIN.EQ.0)THEN
      C=1.0
      S=0.0
```

```
      AOUT=U
      ELSE
C     CALL GAUSS(IX,R,BM,V)
      C=U/(SQRT(U**2.0D0+AIN**2.0D0))
C     C=U/(SQRT(U**2.0D0+AIN**2.0D0))+V
      S=AIN/(SQRT(U**2.0D0+AIN**2.0D0))
C     S=AIN/(SQRT(U**2.0D0+AIN**2.0D0))+V
      AOUT=SQRT(U**2.0D0+AIN**2.0D0)
      ENDIF
      RETURN
      END
C     ******************************************

C     ************INTERNAL CELL***************
      SUBROUTINE INTERNAL (PU,AIN,PC,PS,AOUT,LU,LC,LS)
      REAL PS,PC,PU,S,C,U,AOUT,LU,AIN,LC,LS
      LU=(-PU*PS)+(AIN*PC)
      AOUT=(PU*PC)+(AIN*PS)
      LC=PC
      LS=PS
      RETURN
      END
C     ******************************************


C     *********SUBROUTINE  GROUPS FOR*********
C     ************GAUSSIAN NOISE***************
      SUBROUTINE GAUSS (IX,S,AM,V)
      A=0.0
      DO 90 I=1,12
      CALL RANDU(IX,IY,Y)
      IX=IY
90    A=A+Y
      V=(A-6.0)*S+AM
      RETURN
      END


      SUBROUTINE RANDU (IX,IY,YFL)
      IY=IX*65539
      IF (IY) 5,6,6
5     IY=IY+2147483647+1
6     YFL=IY
      YFL=YFL*.4656613E-9
      RETURN
      END

C     ******************************************
```

LIST OF REFERENCES

1. Goodwin, G. C. and Sin, K. S., *Adaptive Filtering, Prediction and Control,* pp. 49-68, Prentice-Hall, 1984.

2. Ljung, L. and Soderstrom, T. *Theory and Practice of Recursive Identification,* MIT Press, 1983.

3. Kung, H. T. and Leiserson, C. E., *Sparse Matrix Proceedings,* SIAM, 1978.

4. Stewart, G. W., *Introduction to Matrix Computations,* pp. 209-215, Academic Press, 1973.

5. Kim, Yong Hong, *A Parallel Structure for On Line Identification and Adaptive Control,* Masters Thesis, Naval Postgraduate School, Monterey, California, March 1987.

6. Hsia, T. C., *System Identification,* Lexington Books, 1977.

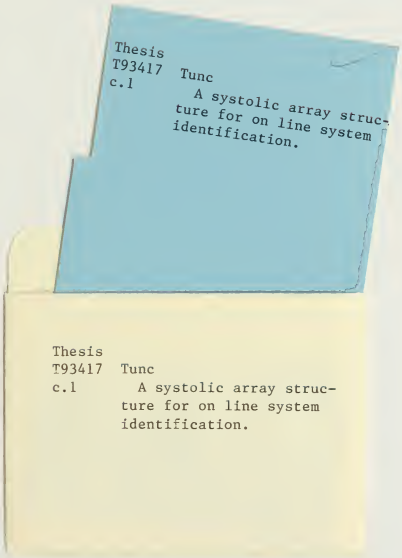
7. Willis, Paul A., *Adaptive Identification by Systolic Arrays,* Masters Thesis, Naval Postgraduate School, Monterey, California, December 1987.

8. Schmid, H., *Decimal Computations,* A Wiley-Interscience Publication, March 1974.

9. Rialan, C. P. and Scharf, L. L., *Cellular Architectures for Implementing Projection Operators,* IEEE Transactions on Acoustics, Speech and Signal Processing, Volume ASSP-35, pp. 1619-1627, November 1987.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Professor Roberto Cristi, Code 62Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	2
5.	Professor C.H.Lee, Code 62Le Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	2
6.	Deniz Kuvvetleri Komutanligi Personel Egitim Daire Baskanligi Bakanliklar-Ankara TURKEY	1
7.	Deniz Harp Okulu Komutanligi Kutuphanesi Tuzla - Istanbul TURKEY	1
8.	Mumtaz Tunc Plevne Caddesi, Girgin Sokak, No=3 Gulveren - Ankara , TURKEY	2
9.	OP - 02 Undersea Warfare Washington, D.C. 20350	1


Thesis
T93417 Tunc
c.1 A systolic array struc-
ture for on line system
identification.
Thesis
T93417 Tunc
c.1 A systolic array struc-
ture for on line system
identification.